JPL D-

Mars Science Laboratory (MSL) Focused Technology Program

2-D Target Tracking Technology Validation Report

Initial Release

Prepared By:
Won S. Kim
Robert C. Steinke
Robert D. Steele

Document Custodian:
Won S. Kim

December 2003

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109-8099

MSL Rover Technology Validation Report: 2-D Target Tracking

Signature Sheet

Approval

Won S. Kim, Task Manager, Instrument Placement Valdiation Task Date

Richard Volpe, Mars Technology Program Office Date

Revision History

Revision	Date	Description	Author
Initial Release	12/23/2003	Initial release	Kim, Steinke, Steele

Related Documents

Document Name	Date	Author
CLARAty Delivery for 2-D Visual Tracking	4/1/2003	Nesnas, Bajracharya

Scope

The scope of this document is to report the results on test and validation of the JPL 2-D visual target tracking software algorithm. This algorithm validation process supports technology selection process before flight qualification of the software selected. The test validation matrix for the 2-D tracker software is shown below. [Table to be inserted].

Acknowledgment

This work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. We would like to thank Issa Nesnas and Max Bajracharya for providing the CLARAty 2-D visual tracking software package and valuable technical discussions.

Contents

Revision History……………………………………………………………………………. 2
Related Documents…………………………………………………………………...…. 2
Scope……………………………………………………………………………..……. 3
Acknowledgment……………………………………………………………………4

Summary……………………………………………………………………………………. 6

1 Introduction……………………………………………………………….……….…… 8

2 2-D Target Tracking Software……………………...……………...……………………. 10

3 Test Plan and Experimental Procedure………………………………….……….……………... 8
3.1 Test Plan…………….………….……….……………………………………….…… 11
3.2 Experimental Control Variables…………..…………………………..……….…………12
3.3 Experimental Setup………...………….…………………………..…….….………. 12
3.4 Experimental Procedure……………….…………………………..…….….………. 14

4 2-D Tracking Performances………………………………………………….……….……. 17
4.1 Window Size and Pyramid Level…………….…………..………….…….………….. 17
4.2 Transforms….…………..……….…….……………………………….……..…..… 20
4.3 Tracking Configuration…………….…………..………….…………….…………..… 24
4.4 Template Window Update……………………….……………………….………...………. 27
4.5 Maximum Image Displacement for Active Camera Control………..…………………. 29
4.6 Lighting Changes……………….………………………….…………………….……………. 31
4.7 Tracking with Oblique and Sideway Camera Motion…………………..……….… 35
4.8 Tracking with Actual Rocky8 Mast Camera Images…………………….…………..… 37
4.9 Average Tracking Performances from Forward, Roll, and Yaw Camera Motion…….. 38
4.10 Tracking with Small-Image-Size Rocks.…………………………………………… 41
4.11 Forward-Motion Tracking with Large-Image-Size Rocks………………………..… 43
4.12 Roll-Motion Tracking with Large-Image-Size Rocks……………………………… 47
4.13 Yaw-Motion Tracking with Large-Image-Size Rocks………………………………50
4.14 Alternate Template Update Method..…………………..…………………………… 55
4.15 Improving Tracking Reliability……………………………………………………. 57

5 Software Bug Findings and Fixes……………...……………….……………………...… 58

6 Conclusion…………………………………………………………………………...……. 60

References……………………………………………………………………………………….. 61

Summary

As part of the Instrument Placement Technology Validation Task, this report details experimental results of 2-D visual target tracking. The 2-D target tracking software which was provided by the Visual Tracking Task at JPL was used in our test and validation within the CLARAty (Coupled Layer Architecture for Robotic Autonomy) software environment. Since 2-D tracking software did not support active camera control, forward motion tests were limited to 4 m. The full 10-m forward motion tests will be performed when the 2-D/3-D tracking software with active camera control is available.

Performance evaluation of 2-D tracking software:

1. **Tracking window size**. The ideal window size was affected by two sources of error. If the window is too small, there may not be enough texture to track well. If the window is too large, the terrain may violate the planarity assumption of the affine tracker and not track well. Experiments indicated that typically 15×15 pixels for small rocks and 29×29 pixels or larger ones for large rocks appeared to be a good size. A larger window size also handles larger image displacements as described next.
2. **Pyramid level**. Large image displacements require a large effective window size to prevent a target feature from leaving the tracking window entirely. As the number of pyramid levels increases by one, the image size is reduced to a half, and the effective window size doubles. For example, a 15×15 window size with 4 pyramid levels has an effective window size of $15 \times 2^3 = 120$ pixels at the top pyramid level. Likewise, 29×29 window size with 3 pyramid levels, a 59×59 window size with 2 pyramid levels, and a 119×119 window size with 1 pyramid level also have similar effective window size in terms of handling large image displacements. Experimental results showed that combinations of widow size and pyramid levels having similar window size yielded similar tracking performance in terms of handling large image displacements.
3. **Transforms and tracking configuration**. 2-D translation tracked more reliably than affine, but drifted more and was less accurate. Affine transform tracked more accurately when it tracked, but lost targets more easily than 2-D translation. Affine transform was good for monitoring tracking. Based on these observations, a new combined tracking configuration was added: 2-D translation followed by affine monitoring/correction. This combined tracking configuration demonstrated good tracking performance.
4. **Template update**. Upper bounds for the template window update interval were about 2 m for straightforward motion at about 10-m target distance (20% change in distance to target) and 20º for roll and yaw motions. However, the optimal update interval depends upon targets and many other factors. Typically, every 50 cm change at 10-m target distance (5% change in distance to target) for forward motion and every 5º to 10 º change in roll or yaw motion appeared to be good template update intervals. Beyond monitoring the distance and orientation changes, it is conceivable to monitor the affine tracker ssd (square sum of differences) threshold as well for template update.
5. **Maximum image displacement for active camera control.** Tracking experiments with various image skips to change image displacement sizes showed that up to about 30 pixels image displacements were tracked for a 15×15 window with 3 pyramid levels.
6. **Lighting variation.** The 9-AM images made dark shadow on the rock surfaces facing the camera. Even though the lighting and shadow condition for 9 AM was significantly poorer than those of 2 PM and 4 PM, the tracking performance for 9 AM was pretty good, demonstrating robustness of tracking relative to the sunlight direction change. In the next experiment, there were dramatic changes in ambient sunlight when an opaque patch of cloud moved across the sun. For 8-mm images, automatic gain control (AGC) was helpful to tracking by maintaining the average intensity of the camera image

relatively constant. By contrast, for 4-mm images, AGC was not helpful due to “bright” sky in view, resulting in 0% tracking. Incidentally, this dramatic lighting change due to dark clouds (opacity > 10) would not happen in Martian environment.

7. **Tracking with Rocky8 Mast Cameras.** Targets tracked well for the first 1.8 m until the rover went over the rock, which caused large image displacements (sometimes over 100 pixels). By 2.4 m, all targets were lost. This clearly indicates that active camera control with 2-D/3-D tracking is essential.
8. **Average tracking performances from forward, roll, and yaw camera motion.** The average tracking performances were above 80% to 100% for all forward, roll, and yaw camera motions. Targets on small rocks tracked 100%, while several targets on large rocks were lost with the 15×15 window size. Forward-motion tests were limited to 4 m, since full 10-m forward-motion tests require active camera control.
9. **Tracking with large-image-size rocks.** Targets on large rocks often did not have enough texture to track well with small windows. When the window size was increased to 29×29 or 59×59, the tracking performance improved to or near 100%. We overlaid affine-transformed template windows on images during tracking, which turned out to be extremely helpful to assess and understand the affine tracking performance.
10. **Improving tracking reliability**. Items include: 1) active camera control with 2-D/3-D tracking, 2) brute-force cross-correlation 2-D translation, 3) template update triggered by the thresholds, 4) off-centered window, 5) selective-region window, and 6) an advanced algorithm that detects target drift and loss reliably.

Software bug findings:
Tracking performance with the initial software delivery was poor. Through collaborative efforts with technology providers, several software bugs were identified. The final software delivery after three critical bug fixes showed dramatic performance improvements. Five major bugs identified including the first three critical ones were listed here.

1. **Partial derivatives inverse matrices at different pyramid levels**. Partial derivatives inverse matrices were not retrieved correctly at lower pyramid levels. This screwed up feature matching at lower pyramid levels.
2. **Delta translation updates between pyramid levels.** The delta translation updates were not halved when the pyramid level for matching went down, and not doubled when it went up. This also screwed up feature matching at lower pyramid levels.
3. **Interchange of width and height.** In two places, image width and height were swapped. This prevented tracking beyond the 768-pixel column of the 1024-column image.
4. **Crash after long runs with a large number of targets.** There are memory leaks in Feature_Window and Image_Pyramid classes. Objects created with new() during class constructions could not be deleted upon class destructions. This bug not fixed yet.
5. **Crash for track windows at the image boundary.** The program crashes when target windows are at the image boundary. This bug not fixed yet.

In this report, experiments results were obtained by using the final 2-D tracking software delivery after the fixes of the first three critical bugs. The last two bugs remained to be fixed, but they should not affect the 2-D tracking experimental results.

1. Introduction

At present, the baseline operation of the Mars Exploration Rover (MER'03) flight mission depicts the state of art technology for instrument placement on Mars. As illustrated in Figure 1, if a target rock is about 10 m away from the rover, the MER baseline operation requires 3 sols to place an instrument on the target position of the rock. After surveying the panoramic images received from the rover, scientists select a target rock to explore that might be 10 m away from the rover. Scientists then decide a waypoint that is about 2 m away from the target, and send a go-to-waypoint command to the rover. The rover moves 8 m, takes images, and sends them to Earth. Using these images, scientists and ground operators determine the rover base or anchor position where the target rock is within arm's reach, and send the corresponding go-to-waypoint command. The rover then travels 2 m reaching the target, takes close-up target images, and sends them to Earth. Scientists and ground operators determine the manipulator motion commands and send them to the rover. The manipulator moves according to the motion commands, placing an instrument on the target position of the rock.

In Mars Science Laboratory (MSL'09) enhanced operation, it is desirable to achieve the entire target approach and instrument placement from 10 m away in a single sol, as shown in Figure 2. Reducing the 3-sol baseline operation to a single-sol operation increases Mars science return significantly. According to an MSL baseline document [Steltzner 2003], 8 to 10 sols are expected to be spent per rock on the average, assuming 3-sol baseline instrument placement. With the enhanced single-sol instrument placement capability, 8 to 10 dols/rock will be reduced to 6 to 8 sols/rock, resulting in 20% to 25% increase in science return.

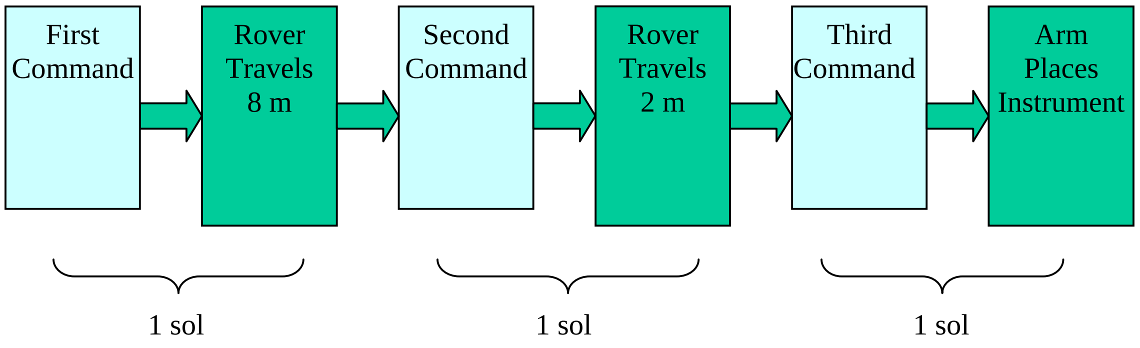


Figure 1. Mars Exploration Rover (MER'03) baseline operation requires 3 sols to place an instrument on a rock from 10 m away

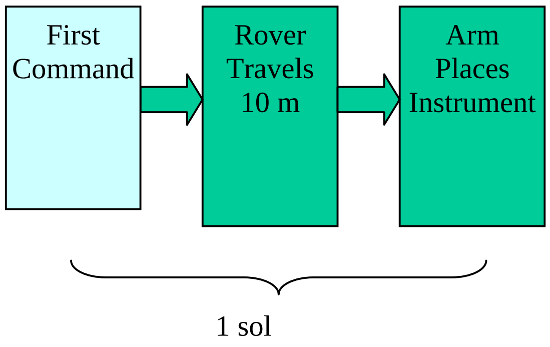


Figure 2. Mars Science Laboratory (MSL'09) enhanced operation requires a single sol to place an instrument on a rock from 10 m away.

The single-sol target approach and instrument placement is technically challenging. In particular, the operation must be fail-safe and reliable. Target approach and instrument placement technologies were demonstrated earlier for some experimental conditions. However, fail-safe, reliable operations have not been demonstrated yet. Extensive experiments are necessary to produce fail-safe, reliable operations for target approach and instrument placement.

Two main technologies to achieve single-sol target approach and instrument placement operations are 1) visual target tracking for approach and 2) rover stereo-based manipulation to place an instrument. Earlier this year, we tested and validated the JPL stereo vision software [Kim, Steinke, Steele, Ansar, 2003] needed for rover stereo based manipulation. In this report we describe the test and validation of 2-D target tracking software with no active camera control. Later we will test and validate the 2-D/3-D tracking software supporting active camera control.

Following the MSL technology infusion process guideline, we tested JPL 2-D target tracking software within the CLARAty (Coupled Layer Architecture for Robotic Autonomy) testbed. CLARAty [Volpe et. al., 2000 & 2001; Nesnas, et. al., 2000; Estlin et. al., 2001] provides a common software environment that enables implementing comprehensive control for planetary rovers and robotic systems.

Section 2 describes the 2-D tracking software, and Section 3 presents the test plan and experimental procedure within the CLARAty environment. Thereafter, Section 4 describes 2-D tracking performance results in detail. Section 5 lists the software bug findings. Finally, Section 6 is Conclusion.

2. 2-D Visual Target Tracker Software

The JPL Visual Tracking Task provided the 2-D visual target tracker within the CLARAty software environment for test and validation. An html document "CLARAty Delivery for 2-D Visual Tracking" [Bajracharya 2003] describes basic software functionalities of the 2-D tracker. It consists of feature detector, feature matcher, template update, image display, and others.

For our test and validation, we added the following several "helper" routines, which are briefly described here.

1. **Input file features_to_add.txt**
   The user can specify initial x and y positions of all targets in the input file named features_to_add.txt. The 2-D tracker software reads the file and recognizes them as initial target positions to track.

2. **Mouse-click user interface for target selection**
   Target positions can also be specified interactively by using a mouse in addition to the text-based entry using features_to_add.txt described above. The user can create and delete targets directly on the displayed image.

3. **Output file posout.txt**
   The 2-D tracker generates the output file named posout.txt, which prints out the target ID, x and y positions for each new image used for tracking.

4. **A combined 2-D translation and affine transform**
   This enables feature matching by a sequential combination of 2-D translation followed by affine transform.

5. **Affine transformed template window**
   We implemented an excellent way to observe how well affine tracking works by displaying the affine transformed windows (quadrilaterals) which are affine transformations of initial template windows (squares). The affine transform does translation, scale, stretch, and shear of the initial template image of a square window in order to match with the current target image. This capability enabled us to visually examine affine tracking behavior.

6. **Optional printouts of feature match parameters**
   The 2-D tracker can optionally print out 2-D translation parameters after each feature match. It can also print out transform parameters for each pyramid level to examine the computational behavior of iterative match in more detail.

3. Test Plan and Experimental Procedure
3.1 Test Plan

Day 1: Collect camera images using a linear motion stage on a sunny day

- Three different sunlight directions
  - Morning (9 am)
  - Early afternoon (2 pm)
  - Late afternoon (4 pm)
- Three different directions of linear motion
  - Straightforward (parallel to the camera axes) for all
  - Oblique (30 degrees to the camera axes) for morning only
  - Sideways (perpendicular to the camera axes) for morning only
- Four lenses mounted on the camera head
  - 16 mm, 17° × 13° [e.g.: linearMotionStage/forward-9am/16mm/*.pgm]
  - 8 mm, 33° × 25° [e.g.: linearMotionStage/oblique-9am/8mm/*.pgm]
  - 4 mm, 65° × 49° [e.g.: linearMotionStage/sideways-9am/4mm/*.pgm]
  - 2.3 mm, 113° × 86° [e.g.: linearMotionStage/forward-2pm/2.3mm/*.pgm]
- Collect hi-resolution camera images every 1 cm over 4 m for all above conditions
  - High resolution CCD images: 1024 pixels × 768 pixels
  - File examples: Hi-A-40.pgm to Hi-A-440.pgm (401 images)

Day 2: Collect camera images using a linear motion stage on a cloudy day

- Clouds moving across the sun, causing significant change in lighting condition
  - Shadow contrast changes as clouds move across the sun
  - Late afternoon (5 pm, cloudy)
- Straight forward direction only
- Four lenses mounted on the camera head
  - 16 mm, 17° × 13° [e.g.: linearMotionStage/forward-5pmCloudy/16mm/*.pgm]
  - 8 mm, 33° × 25° [e.g.: linearMotionStage/forward-5pmCloudy/8mm/*.pgm]
  - 4 mm, 65° × 49° [e.g.: linearMotionStage/forward-5pmCloudy/4mm/*.pgm]
  - 2.3 mm, 113° × 86° [e.g.: linearMotionStage/forward-5pmCloudy/2.3mm/*.pgm]
- Collect hi-resolution camera images every 1 cm over 4 m

Day 3: Collect camera images for different roll and yaw orientations on a sunny day

- One sunlight direction: Early afternoon (2 pm)
- For roll:
  - Collect images for every 1 degree roll of the tripod from –45 to +45 degrees
  - File example: linearMotionStage/roll-11am/16mm/*[0-90].pgm
- For yaw:
  - Collect images for every 2 cm on a linear motion stage placed sideways over 4 m
  - Re-orient the camera head to point to a target located 5 m straight in front
  - File example: linearMotionStage/yaw-3pm/8mm/*[0-200].pgm

Day 4: Collect camera images using the mast on Rocky8 on a sunny day

- One sunlight direction: Early afternoon (2 pm)
- Straight forward direction only
- PanCam and NavCam pairs mounted on the mast
  - 16 mm, 17° × 13° [e.g.: rocky8Live/forward-2pm/16mm/*.pgm]
  - 4 mm, 65° × 49° [e.g.: rocky8Live/forward-2pm/4mm/*.pgm]
- Collect hi-resolution camera images every 1 cm over 10 m

3.2 Experimental control variables

- Window size
    - 7, 15, 29, 59, 75, 119
- Number of pyramid levels
    - 1, 2, 3, 4, 5
- Template window update interval
    - 1, 10, 20, 100, …
- Transformation
    - 2-D translation, scale, z_rotation, affine transform
- Image skip
    - 0, 1, 2, 3, 5, 10, 20, …
- Camera motion
    - Straightforward, oblique at 30 degrees, and sideways
    - Roll
    - Yaw
- Sunlight direction
    - The sun in front of, above, or behind the camera
- Lighting variation
    - Abrupt changes in sunlight as an opaque clouds pass across the sun
- Target selection to track
    - Different texture
    - Rock image sizes
    - Occluding boundaries

3.3 Experimental Setup

Four high-resolution firewire Dragonfly cameras manufactured by Point Grey Research were used for the experiments. The resolution was 1024 pixels × 768 pixels. Four kinds of lenses were used in the experiment: 2.3 mm, 4 mm, 8 mm, and 16 mm lenses for a 1/3-in CCD image format. Table 1 summarizes their field of view angles for the lenses used.

Focal length	Horizontal FOV × Vertical FOV
Computar 2.3 mm	113° × 86°
Fujinon 4 mm	65° × 49°
Fujnon 8 mm	33° × 25°
Fujinon 16 mm	17° × 13°

Table 1. Lenses used for the 2-D target tracking experiments

The images were collected using a laptop computer with an OrangeLink firewire card-bus PC card. The Point Gray's Dragonfly image capture software that had been modified previously for the camera calibration and stereo vision experiments, to support the consecutive acquisition of camera images with new filenames, was again used for the 2-D target tracking experiments. A firewire hub was used to allow up to 5 cameras to be connected to the computer simultaneously.

We designed a wooden linear motion stage (Figure 3), and the JPL carpenter shop built it very inexpensively. The linear motion stage was necessary to move the camera head by a small increment over a long distance, avoiding large image displacements between consecutive images. As an example, we collected camera images every 1 cm over 4 m using this linear motion stage.

The use of the linear motion stage was essential due to the lack of active camera control of the 2-D tracker.

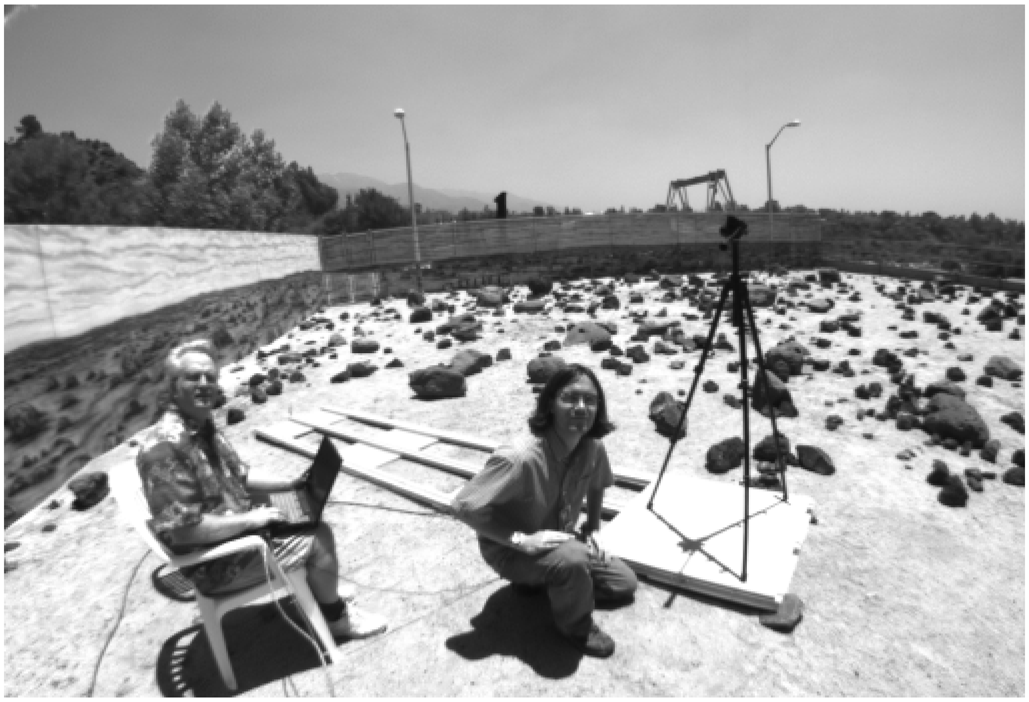


Figure 3. A wooden linear motion stage used in the Mars Yard to move the camera head linearly by small increments and collect image sequences. The camera head was mounted on the tripod.

In the initial 2-D tracker software, the computer automatically selected targets to track based on corner detector scores. Since we needed to designate targets anywhere we wanted for the test and validation of the 2-D tracking algorithm, we added a user interface that allows the user to enter the target positions by mouse click. The following was the mouse-click user interface design specification we came up with. Upon running the 2-D tracking program, the initial image will be displayed for the user to select or specify targets via the user interface.

The two modes of the user interface are:

- Point-click mode [User-specified target creation/selection mode] and
- Region selection mode [Computer-generated target creation/selection mode].

The steps for the user-specified target creation/selection mode are:

- Double-click the left mouse button outside of any existing targets to specify the target position. One click will request the user to click one more time. If the two click positions are farther apart than the 2-D track window size, the user interface goes to the region selection or computer-generated target creation/selection mode described below.
- The goodness value for this target will be displayed.
- To reject a target position, click the target with the right mouse button.
- To accept this target, press the left mouse button.
- To select an existing target, click the target with the left mouse button. The goodness values of the target will be displayed. To accept the target, click the left mouse button again to confirm it. To reject the target, click the right mouse button.

- To reject an existing target without selecting it, click the target with the right mouse button.
- To exit this target designation user interface, click the right mouse button outside any exiting targets. A list of goodness values for all targets will be displayed. To confirm the exit, click the right mouse button again. To cancel the exit, click the left mouse button.

The steps for the computer-generated target creation/selection mode are:

- Click the two diagonal endpoints of a region with the left mouse button to indicate the region. If the region is smaller than the 2-D track window size, the user interface goes to the point-click or user-specified target creation/selection mode described above.
- The 2-D-Tracker software will find targets in this region, and these computer-generated targets will be displayed to the user.
- To select a target, click the target with a left mouse button. The goodness values of the target will be displayed. To accept the target position, click the left mouse button again to confirm it. To reject a target position, click the target with the right mouse button.
- To reject a target position without selecting it, click the target with the right mouse button.

To exit the region selection mode and go back to the user-clicked target creation/selection mode, click the mouse outside the region.

Although we implemented the above user interface, we ended up mostly not using it because computer-selected targets were not that useful at present, generating too many undesirable targets. This is because the corner detector of the 2-D tracker computes the goodness value per pixel, not per window, by considering only a very small local area independent of the window size. Instead, we implemented a much simpler text-based target entry using features_to_add.txt, which turned out to be very convenient in our experiments because it remembers the initial target positions without requiring the repeated entry of the same initial target positions. It was quite difficult and time-consuming to enter exactly same initial target positions using the mouse-click interface.

3.4 Experimental Procedure

Here is the procedure on how to download, compile, and run run_feature_tracker.

To check out the CLARAty stereo vision package from the CLARAty repository,

```
tcsh                    // csh causes an error
klog user_name          // enable access of afs files
start_claraty           // set environmental variables
yam setup -nolink -nobuild -d dir_name
```

When YAM.config comes up, edit it as shown below, or just copy and paste The content of WORK_MODULES, LINK_MODULES, and BRANCH* from /home/marstech/IPvalidation/2dtracking/claraty/*/YAM.config.

```
WORK_MODULES =    corner_detect_op \
                  feature_tracker \
                  image_displayer
```

```
LINK_MODULES =    SiteDefs/SiteDefs-R1-10l \
                  arrays/arrays-R1-08b \
                  data_io/data_io-R1-00d \
                  draw_ops/draw_ops-R1-02 \
                  image/image-R1-04c \
                  image_io/image_io-R1-03b-Build01 \
                  image_io_pnm/image_io_pnm-R1-01 \
                  image_ops/image_ops-R1-04-Build02 \
                  image_pyramid/image_pyramid-R1-01 \
                  image_rgb/image_rgb-R1-01-Build02 \
                  matrices/matrices-R1-09b \
                  points/points-R1-08a \
                  share/share-R1-09a \
                  string_io/string_io-R1-02c

BRANCH_corner_detect_op   =   corner_detect_op-R1-01  steele
BRANCH_feature_tracker    =   feature_tracker-R1-01   steele
BRANCH_image_displayer    =   image_displayer-R1-03   steele
```

To compile:

```
cd <sandbox>    // <sandbox> is dir_name specified in the “yam setup” command above
gmake all
cd src/feature_tracker
ymk all         // compile feature_tracker source codes to create run_feature_tracker
```

A couple of source codes, feature_tracker_app.cc and feature_matcher_lk.h, can be compiled with different compiler options by either commenting or un-commenting define statements.

In feature_tracker_app.cc:

#define NewTwoClickInterface	This option allows mouse click entry of target positions.
#define ignoreBobSteinkeAffineCode	If not defined, a combined 2-D translation and affine transform is used in matching
#define AFFINE_POINTS	Computes affine transformed template windows and superimposes the computed quadrilaterals on the image
#define noAffineTemplateUpdate	Affine template is not updated while 2-D translation template is updated

In feature_matcher_lk.h:

#define REJECT_LARGE_DELTA	This option allows rejecting target positions if the iterative algorithm generates unexpected large displacements for next target positions
#define PRINT_DELTA_T	Prints the translation parameters tx and ty after the completion of feature matches over all pyramid levels
#define PRINT_V	Prints the transform parameter vector v for each pyramid level

To run run_feature_tracker using Drun:

```
~/<sandbox>/bin/Drun    run_feature_tracker    params.txt
```

Drun allows the user to use correct shared libraries by modifying LD_LIBRARY_PATH to point at <sandbox>/lib/$YAM_TARGET and PATH to point at <sandbox>/bin/$YAM_TARGET.

To run run_feature_tracker without using Drun:
First set LD_LIBRARY_PATH to point at <sandbox>/lib/$YAM_TARGET.

```
run_feature_tracker     params.txt
```

The input file params.txt specifies the input parameters for tracking. An example file is shown here.

params.txt:
```
{
      feature_window_size = 29;
      feature_distance_threshold = 29;
      use_fixed_detect_threshold = 1;
      detector_threshold = 4000;
      detector_percentage_of_max = 75;
      detector_window_size = 3;
      detector_derivative_size = 7;
      use_fixed_match_threshold = 1;
      matcher_max_num_iters = 20;
      matcher_ssd_diff_threshold = 0.1;
      num_pyramid_levels = 2;
      base_filename = "/home/marstech/IPvalidation/
            2dtracking/linearMotionStage/
            roll-11am-2003-08-21/16mm/Hi-C-%03d.pgm";
      window_update_rate = 100;
      max_num_of_images = 100;
      min_num_of_features = 1;
      debug_sleep = 0.1;
      image_num_begin = 0;
      image_num_end = 91;
      image_num_skip = 0;
}
```

In addition to params.txt, run_feature_tracker checks to see if the following input file exists.

features_to_add.txt:	This file specifies x and y image coordinates of each target position per line. This input file has the same effect of the user entering all the listed target points with a mouse.

After the run, run_feature_tracker produces the following outputs.

posout.txt:	each line prints out the image number followed by all features containing feature number, feature position x, and feature position y

4. 2-D Tracking Performances

4.1 Window Size and Pyramid Level

We investigated the effect of window size and pyramid level on 2-D tracking performance. Figure 4 shows the beginning and end images of the image sequence collected for a 4-m straightforward traverse with a 16 mm lens. The image sequence consisted of total 400 images with a 1-cm straightforward motion between consecutive images. We selected 11 targets in the beginning image based on corner detector values. These targets were high-texture targets, and all the targets tracked 100% over a wide range of the window sizes and pyramid levels. In this initial test, no images were skipped, resulting in a 1-cm camera motion between images. In order to compare various window sizes and pyramid levels, we increased camera motion by skipping images until tracking performance was adequately degraded. We chose 19-cm camera motion between images by running the 2-D tracker on every 19-th image to perform multiple runs to compare various window sizes and pyramid levels. Table 2 shows the results of those runs. Each entry shows the number of targets successfully tracked out of 11 together with the drift error in pixels of successfully tracked targets compared to human tracking. Doubling the tracking window size (adjusted to be an odd number) showed a similar effect of reducing the number of pyramid levels by one in terms of tracking performance. Namely, each upward diagonal in Table 2 shows similar tracking performance. In particular, the diagonal from lower-left to upper right corner (green color in Table 2) shows best tracking performance.

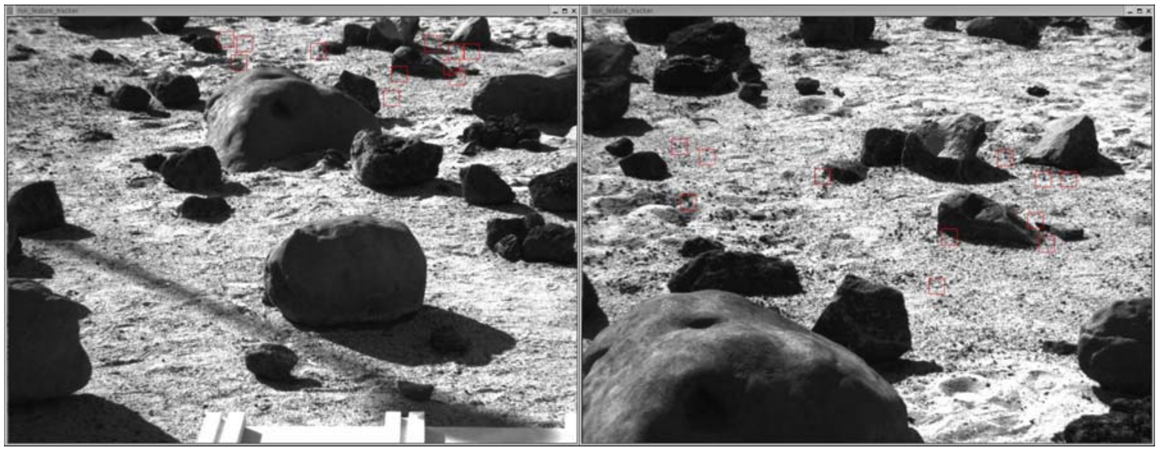


Figure 4: Beginning and end images for a 4-meter straightforward traverse with 16 mm lens, where the 2-D tracker corner detector selected high-texture targets.

Pyramid levels	Window size			
	7x7	15x15	29x29	59x59
1	0/11	2/11 11.6 pixels error	3/11 14.4 pixels error	4/11 15.1 pixels error
2	3/11 11.7 pixels error	5/11 9.6 pixels error	7/11 2.6 pixels error	
3	6/11 12.3 pixels error	10/11 2.0 pixels error		
4	11/11 2.1 pixels error			

Table 2. 2-D tracking success rate and position error results for relatively large 19-cm camera motion between images. Diagonal entries have similar tracking performance.

Increasing the pyramid level by one reduces the image size to a half, and thus doubling the effective window size. If the window size = W pixels and the number of pyramid levels = L, the effective window size = $W \cdot 2^{L-1}$ pixels. The effective window size roughly determines the maximum image displacement that can be tracked. The green-colored diagonal in Table 2 showed the best tracking performance among the diagonals, because it had the largest effective window size (approximately 60 pixel), which certainly helped track targets of large image displacements caused by a relatively large 19-cm camera motion between consecutive images.

In Table 2, all 11 selected targets had good corner features to track. In this case, a small window of 7×7 with 4 pyramid levels performed best with low drift error. On the other hand, the large window of 59×59 performed poorly. The large window sizes tend to include adjacent occluding or occluded objects, which confuse tracking because their relative positions in the image keep changing during the tracking. Further, they violate to a greater degree the assumption of the affine tracker that the target is a planar surface. As long as a 7×7 window has sufficient texture to track, 7×7 window with 4 pyramid levels tracks more accurately than 59×59 with 1 level while the maximum image displacement to track is about the same.

A smaller size window with a large number of pyramid levels starts with a large-area coarse matching at the highest pyramid level, and finishes with fine matching at the lowest pyramid level of 0 with full image resolution. The 7×7 window with 4 pyramid levels starts with a large-area coarse matching at the highest pyramid level of 3 (effective window size = 56 pixels), and finishes with a local fine matching at the lowest level 0 (effective window size = 7 pixels). By contrast, 59×59 window with 1 level does only a large-area matching (effective window size = 59 pixels). The computational efficiency is higher with a smaller window size with a large number of pyramid levels. For example, the window size of 59×59 at 1 pyramid level must compute 59×59 pixels, while window size of 7×7 with 4 pyramid levels computes 4×7×7 pixels.

Next, we investigated tracking on low-texture targets. Figure 5 shows the beginning and end images for a 3-m straightforward traverse with 8-mm camera lens. We selected 25 targets in an arbitrary grid on a large rock with low texture. We then performed similar runs as above but with 5-cm and 10-cm camera motions between images, using the image skip of 4 and 9, respectively. Smaller camera motions were used because the low-texture targets were harder to track. Tables 3 and 4 show the results of these runs. Each entry shows the number of targets successfully tracked out of 25. We did not compute the drift error because with the arbitrary placement of targets and low texture we did not feel human tracking was sufficiently accurate for a meaningful measurement. Instead, a target was considered lost if it moved from its relative position in the grid. In Tables 3 and 4, window size of 29 performed best unlike in Table 2 where 7×7 window performed best. The results indicate that targets with lower texture need to have a larger window size. The 59×59 windows tend to include adjacent occluding or occluded objects whose positions relative to the large rock keep changing during the course of tracking. Some of the targets near the boundaries of the large rock get confused and lost during tracking.

As in Table 2 of the previous high-texture target tracking, the trend of improving performance with increasing pyramid level continues in Tables 3 and 4, but it exhibits a plateau beyond which there is no performance improvement. Increasing the number of pyramid levels increases the effective window size and thus helps to handle larger image displacements. If the number of pyramid levels is sufficiently large to handle largest image displacements occurring, the tracking performance will approach its plateau. The green-colored areas in Tables 3 and 4 show this plateau effect. The plateau for each window size in Table 4 is reached about 1 pyramid level lower than that in Table 3, where image displacements from 5-cm camera motion between images are roughly half of those from 10-cm camera motion.

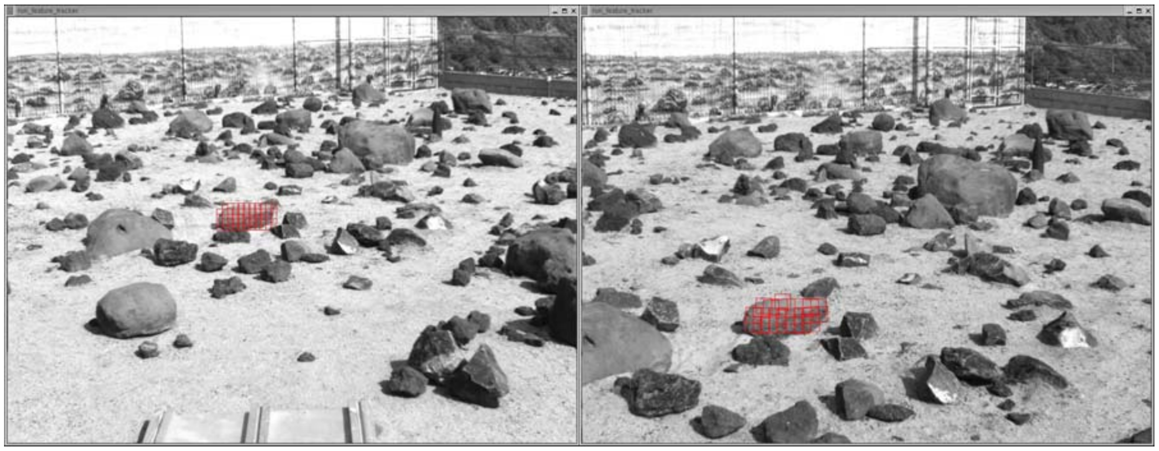


Figure 5. Beginning and end images of 3-m straightforward traverse with 8 mm lens, where a human operator selected a grid of low-texture targets from a large rock.

Pyramid levels	Window size			
	7x7	15x15	29x29	59x59
1	0/25	4/25	11/25	13/25
2	2/25	12/25	17/25	17/25
3	2/25	16/25	21/25	17/25
4	3/25	18/25	21/25	

Table 3. Tracking results for 10-cm camera motion between images (image skip of 9). Green area shows no more improvement in tracking performance with higher pyramid level.

Pyramid levels	Window size			
	7x7	15x15	29x29	59x59
1	1/25	10/25	17/25	19/25
2	4/25	18/25	22/25	19/25
3	4/25	20/25	24/25	19/25
4	6/25	20/25	24/25	

Table 4. Tracking results for 5-cm camera motion between images (image skip of 4). Green area shows no more improvement in tracking performance with higher pyramid level.

The tracking results with high-texture and low-texture targets of Tables 2, 3, and 4 indicate two important effects of window size. Non-planarity of the target, in particular, that has occluding or occluded objects, reduces tracking performance for large window sizes. For small-image-size rocks with high texture, small window would be fine. For medium to large-image-size rocks, the window size needs to be appropriately increased to cope with less texture. At the same time, the window size should not be too large to keep the target reasonably planar and not to include adjacent occluding or occluded objects. Therefore, we recommend varying the window size depending on the planarity and texture of the target. This will require that the command generation interface allow the specification of window size at the time a feature is selected.

4.2 Transforms

The 2-D visual target tracker supports four options of transforms to match the template image of a feature to the current image: 2-D translation, z-rotation, scale, and affine transforms. The 2-D translation uses 2 parameters, tx and ty, that shifts the feature in x and y coordinates to find the matching location of the feature in the new image. The scale transform uses a scale parameter in addition to 2 translation parameters. The scale parameter resizes the feature template image window for matching. The z-rotation transform uses a z-rotation or roll parameter in addition to 2 translation parameters. The z-rotation parameter rotates the feature template image window in roll orientation for matching. The affine transform uses 6 parameters: 4 deformation parameters in addition to 2 translation parameters. The four deformation parameters are the elements of a 2×2 deformation matrix D that allows scaling, rotation, stretch, and shear of the 2-D planar template image window, assuming the feature can be approximated to be planar.

These four transforms were applied to three sets of the image sequences collected earlier with 3 different camera motions of 4-m straightforward, 90º roll, and 45º yaw. Only high-texture targets with salient corner features were selected. No template updates were used in these tests. Namely, the feature windows selected at the beginning images were used as template images all the way to the end images. For the forward (Figure 6) and yaw (Figure 7) camera motions, 2-D translation performed best, while affine transform did worst. Since the yaw camera motion images often had relatively large image displacements between consecutive images, pure yaw tracking tests were also performed by providing correct 2-D translation parameters as an initial seed for each new image matching. The two tracking performances, however, turned out to be similar. For the roll camera motion (Figure 8), z-rotation and affine transforms performed best, while scale transform did worst. Camera roll motion had the same effect of simply rotating the 2-D images, which is essentially 2-D planar transformation regardless of the 3-D shapes of objects in the scene. Ideally, this allows perfect matches for both z-rotation transform and affine transforms. For this reason, z-rotation and affine transforms performed better than 2-D translation for the roll motion. Unlike the roll camera motion, forward and yaw camera motions generally involve 3-D transformation, and in these conditions 2-D translation resulted in more reliable tracking.

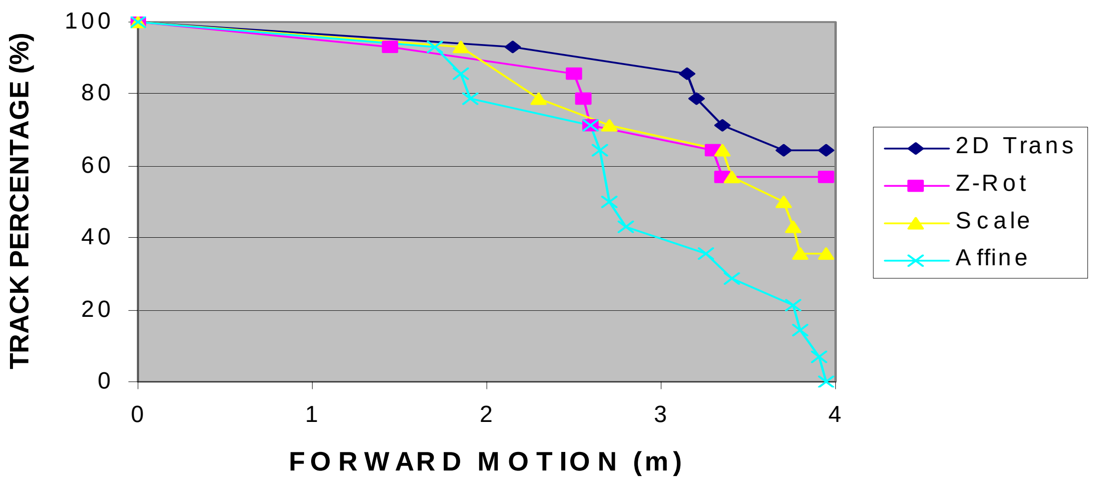
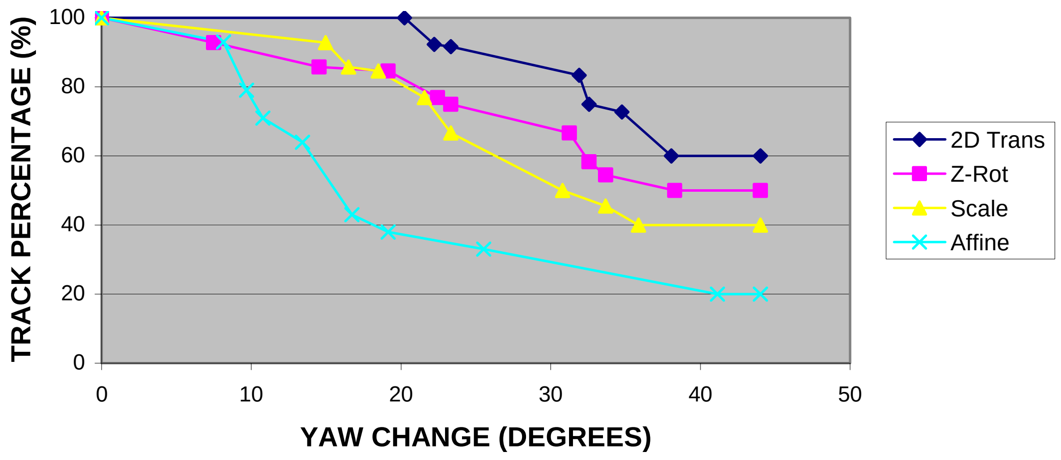


Figure 6. Tracking performance of different transforms over 4-m straightforward motion (no template image update).



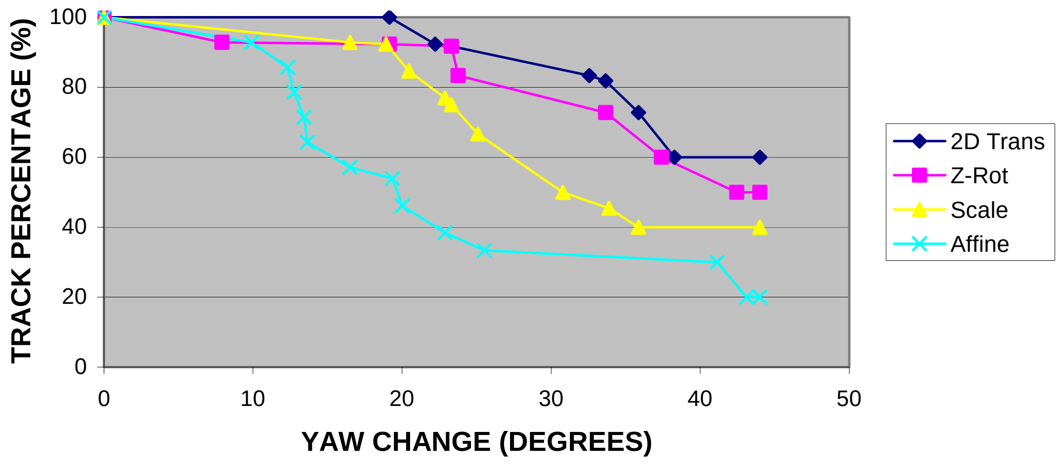


Figure 7. Tracking performance of different transforms over 45° yaw motion: yaw mixed with translation motion (top) and yaw only motion by providing corrective initial seeds for 2-D translation parameters (bottom).

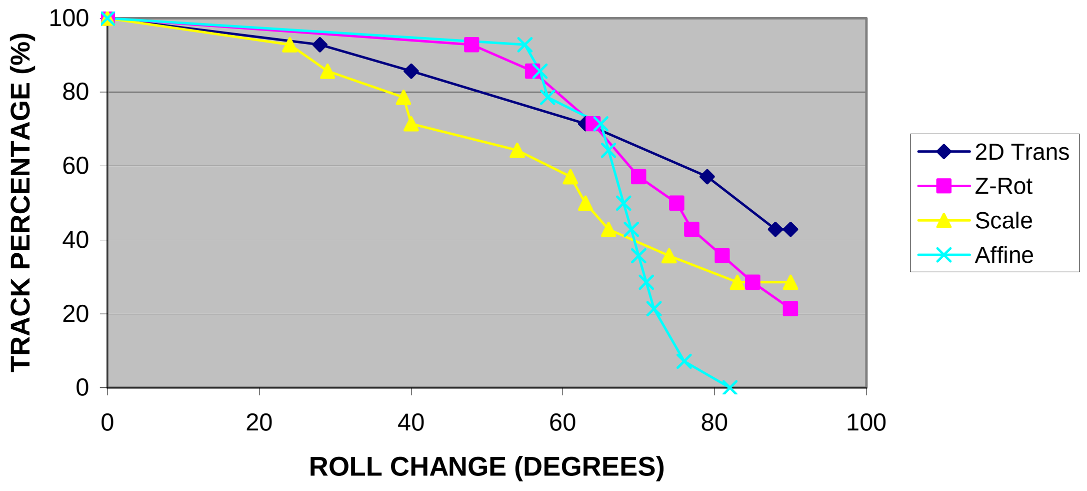
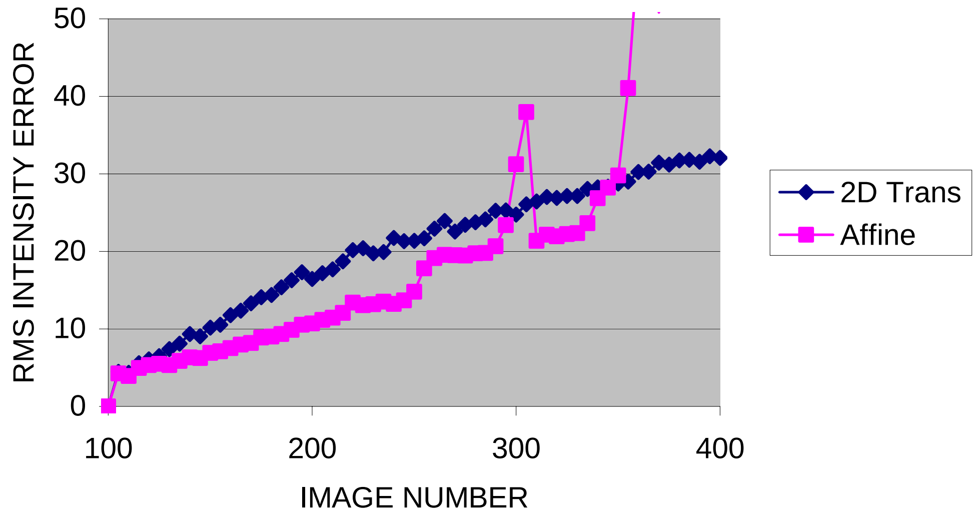


Figure 8. Tracking performance of different transforms over 90° roll camera motion

Figure 9 shows the rms intensity matching errors of 2-D translation and affine transform during the course of 2-D tracking of an example target over the 3-m forward camera motion. The rms intensity error is the per-pixel normalized ssd intensity error between the template image window and the matching current image window, where 255 is the maximum intensity error. When the template image was not updated (Figure 9 top), the rms intensity error increased with the image number for both 2-D translation and affine transform. The affine transform yielded slightly lower rms intensity error when it tracked. However, it lost tracking (at image number=350), while 2-D translation never lost the target. When the template image was updated (Figure 9 bottom) for every 100 images, the rms intensity error did not increase high, and the rms error dropped sharply at each template update. With template update, both 2-D translation and affine transform did not lose the target, demonstrating that template update improves tracking performance.



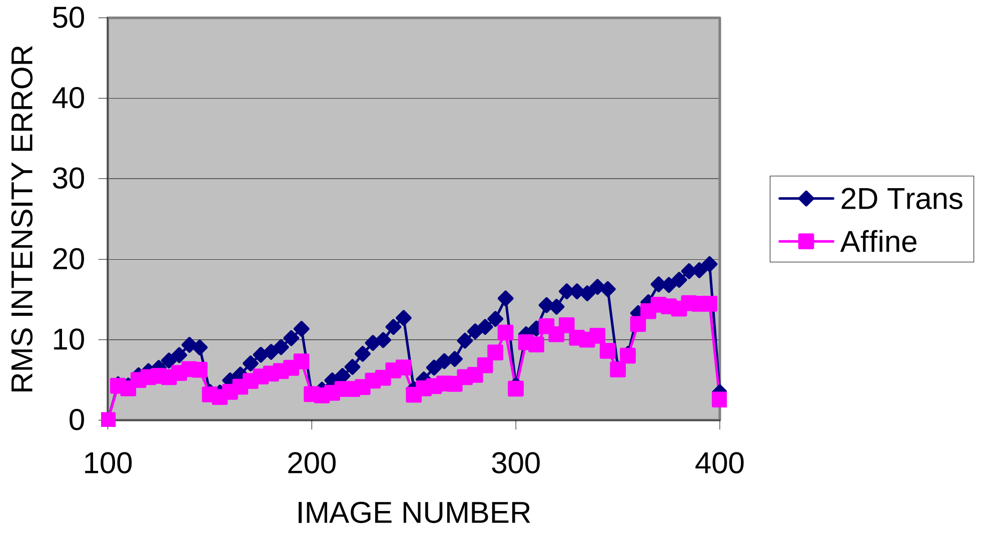
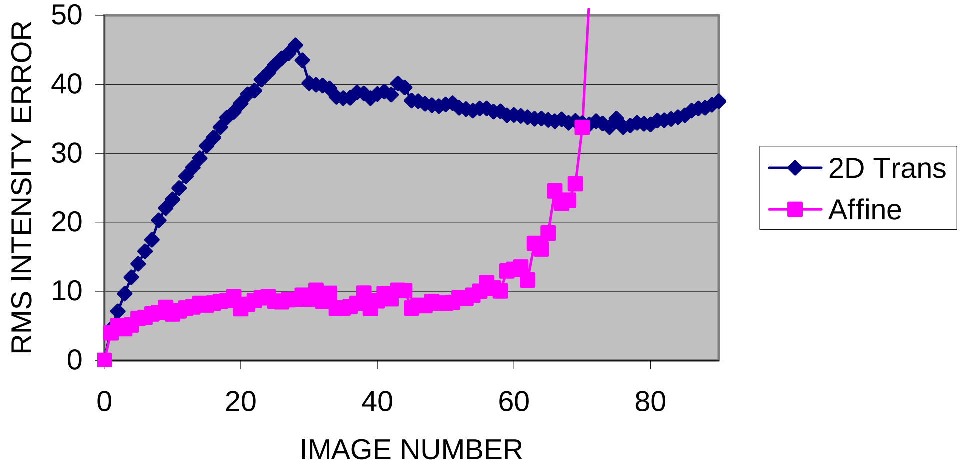


Figure 9. The rms intensity errors of 2-D translation and affine transform over 3-m straightforward camera motion from image number 100 to 400. No template update (top) and template update of every 100 images or 1 m (bottom).

Similarly, Figure 10 shows the rms intensity matching errors of 2-D translation and affine transform during the course of 2-D tracking of an example target over the 90º-roll camera motion. When the template image was not updated (Figure 10 top), the affine transform yielded significantly lower rms intensity error when it tracked. However, the affine transform lost tracking (at image number=70), while 2-D translation never lost the target. This example clearly illustrates that affine transform is more accurate, but less reliable. When the template image was updated (Figure 9 bottom) for every 20 images or 20º roll change, the rms error dropped sharply at each template update. With template update, both 2-D translation and affine transform maintained low rms intensity error and did not lose the target.

Our initial experiments suggested that 2-D translation transform resulted in the most reliable tracking. However, translation often suffered from large drift errors. Affine transform performed best with smaller drift errors when it tracked, but was not as reliable as 2-D translation transform, losing targets more easily. More comparative experiments are presented in the next Section.



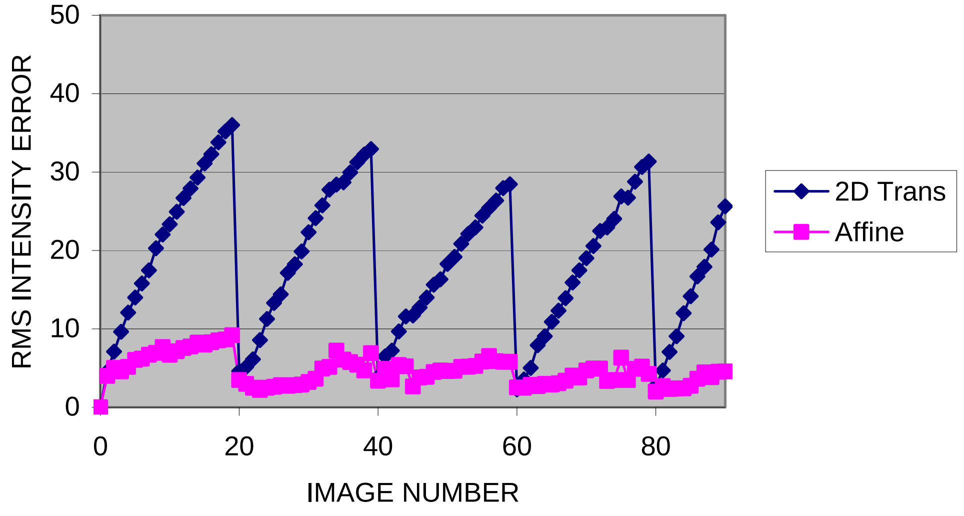


Figure 10. The rms intensity errors of 2-D translation and affine transform over 90° roll camera motion from image number 0 to 90; no template update (top) and template update of every 20 images or 20° (bottom).

4.3 Tracking Configuration

Based on the initial experimental results described in section 4.2, we decided to use a combined configuration of the 2-D translation matching and affine matching to maximize performance (Figure 11). 2-D translation has the best reliability while affine transform has the best accuracy. Therefore, we first use the 2-D translation matching to determine an approximate match the feature. This (tx1, ty1) position is then fed into the affine transform matching to create a more accurate match if possible. The ssd (square sum of differences) error is a good indicator of whether the affine matcher was successful. Therefore, the affine position result is only used if its ssd error is less than the translation ssd error. Otherwise the translation position is used.

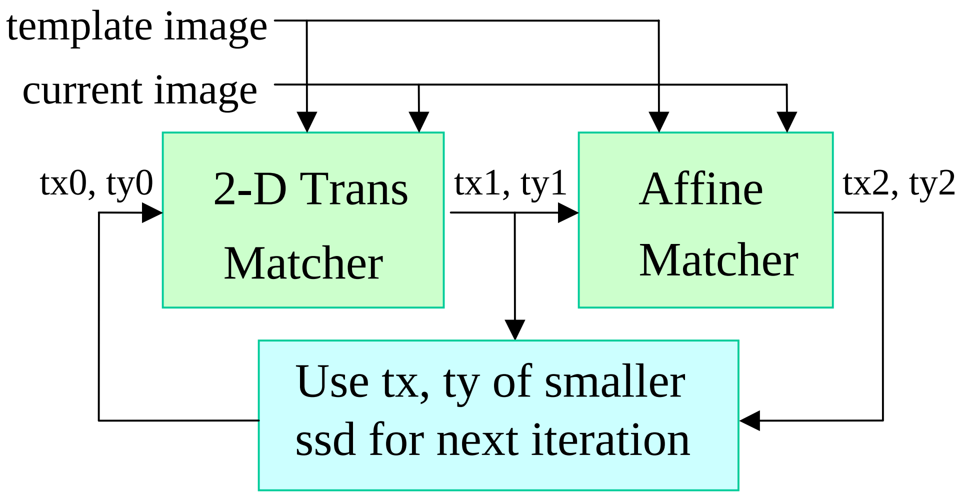


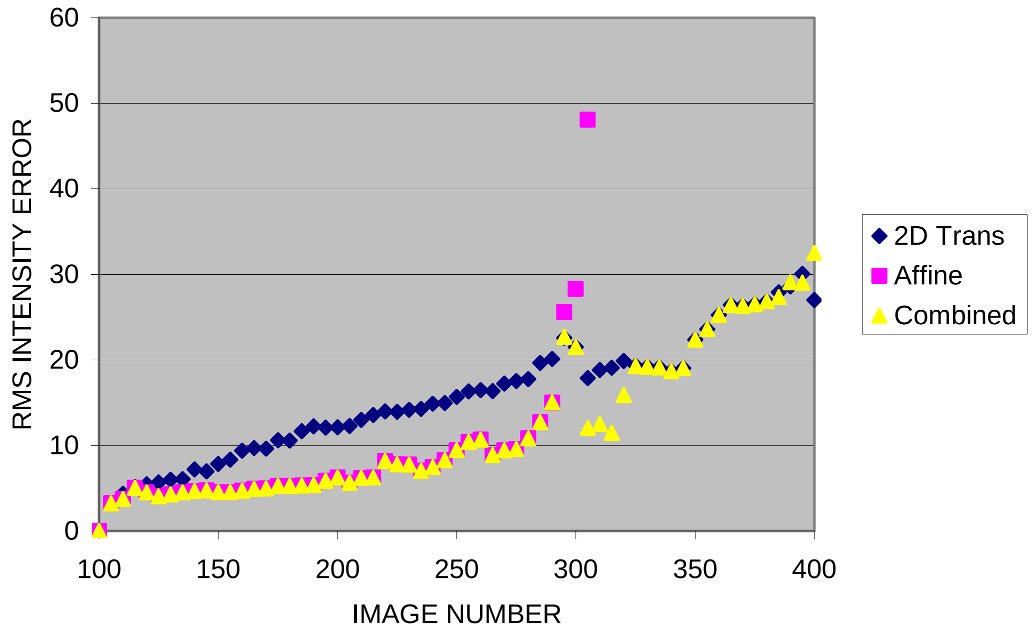
Figure 11. A combined tracking configuration: 2-D translation matching followed by affine matching.

In order to compare the three methods of 2-D translation alone, affine transform alone, and the combined configuration, we performed 2-D tracking experiments on an image sequence of straightforward camera motion with an 8-mm lens. The template update interval was set to 10 or every 10-th image. The tracking results are shown in Table 5. Affine transform alone was less reliable: 49% tracking for 15×15 window with 3 pyramid levels and 89% for 29×29 window with 2 pyramid levels. By contrast, the 2-D translation alone and the combined configuration were more reliable: 98% to 100%.

	15×15 window with 3 pyramid levels	29×29 window with 2 pyramid levels
2-D translation alone	44/45 (98%)	44/45 (98%)
Affine transform alone	22/45 (49%)	40/45 (89%)
Combined configuration	44/45 (898%)	45/45 (100%)

Table 5. Tracking success rates of 2-D translation, affine, and combined configuration for straightforward camera motion with 8-mm lens

Tracking behaviors of 2-D translation, affine, and combined configuration for two targets are compared in Figure 12. For better comparison, no template image was updated, and two targets were chosen that were not tracked all the way by the affine tracker. The rms intensity error (normalized ssd per pixel) of the affine matching started lower, but eventually the affine matching lost the target. On the other hand, 2-D translation tracking had higher rms intensity error, but did not lose the target. Combined tracking basically chooses the better of the two matching results. This method effectively combines the strengths of both transforms. Initially, the rms error of the affine transform stays low providing an accurate position estimate, and when the affine tracker loses the target, the 2-D translation matching maintains the approximate position of the target to bring the affine tracker back on target.



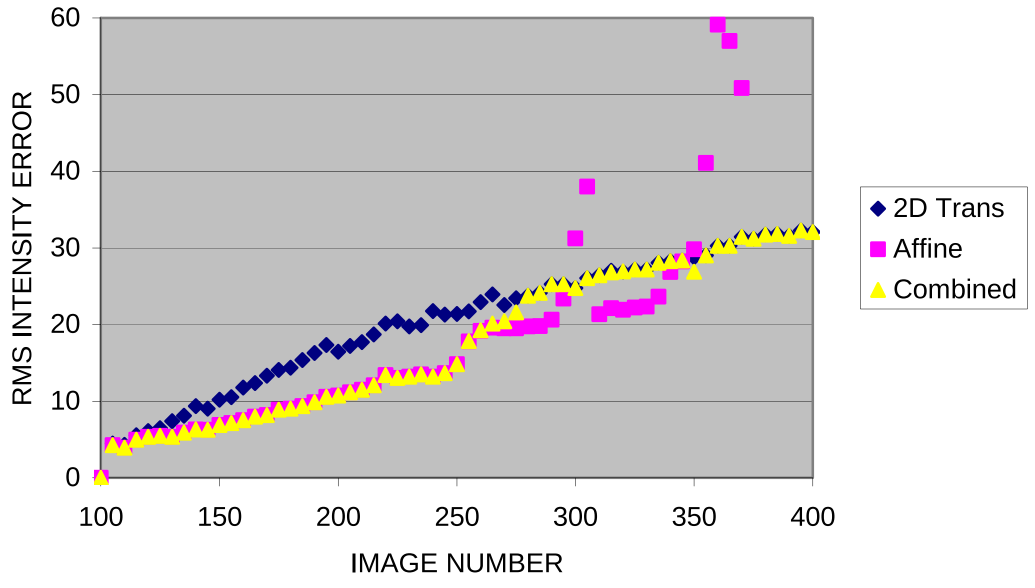


Figure 12. Tracking intensity errors as a function of the image number for 2-D translation, affine, and combined configuration for two targets (top and bottom).

Figures 13, 14,and 15 compare tracking results of three different transform methods. When 2-D translation matching with no template update is used, target location shifted from the left corner to the right corner of the bigger rock at the end of the track (Figure 13). When the template is updated every 20º, the target location still shifted (Figure 14), but not as much as no template update of Figure 13. When the combined 2-D translation and affine transform was used, the target tracked very well (Figure 15).

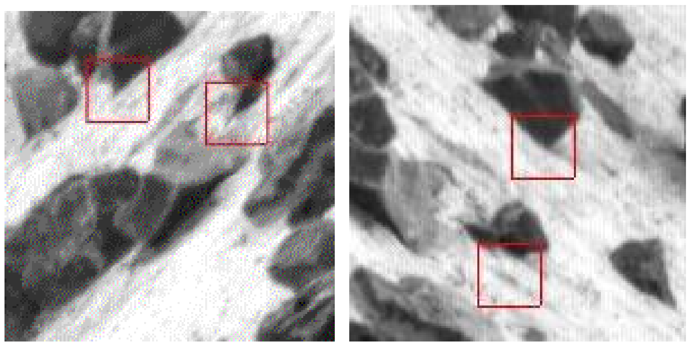


Figure 13. 2-D translation only; No template update over 90º
Tracking error: 30 and 39 pixels

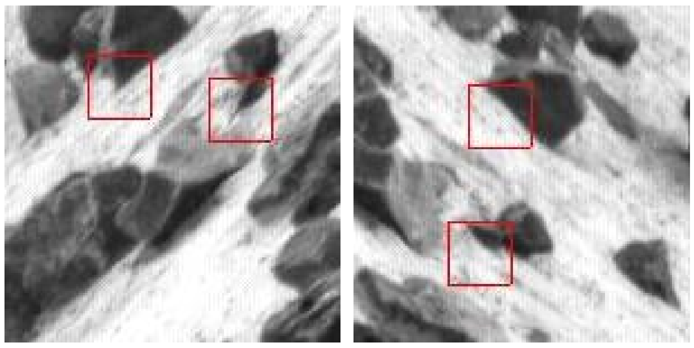


Figure 14. 2-D translation only; Template update every 20º
Tracking error: 13 and 24 pixels

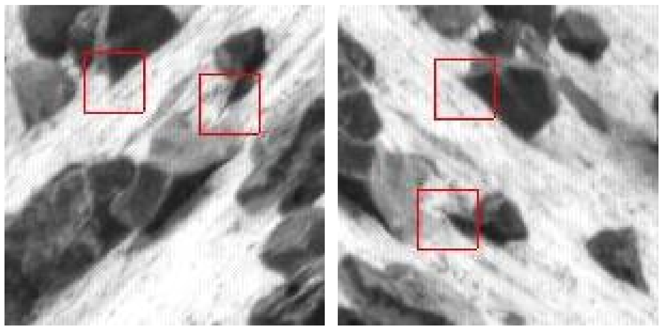


Figure 15. 2-D translation with affine correction; Template update every 20º
Tracking error: 1.7 and 1.0 pixel

4.4 Template Window Update

The current version of 2-D tracker updates the template image window for each target periodically with the update interval controllable as an input parameter. Previous experiments in Figures 6, 7, and 8 provide rough upper bounds for the template window update: 2 m interval for 4-m straightforward motion starting from about 10 m away from the target position (20% change in distance), and 20º change for roll and yaw motions. Beyond these upper bounds, the tracker starts losing targets due to a large change between the template image and the current image. To examine the effect of the template window update on actual 2-D tracking performance, we ran several experiments varying the template update parameter. Figures 15 and 16 show the results from two such experiments.

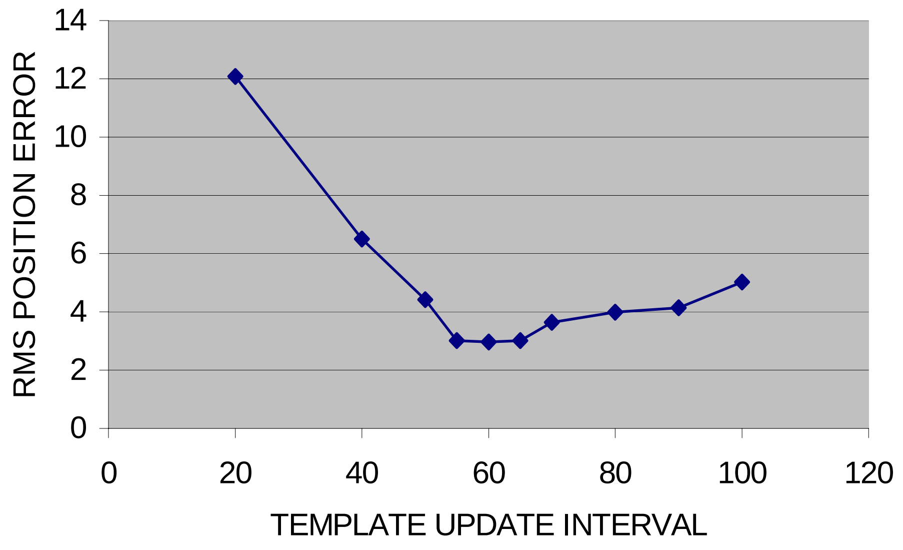


Figure 16. Tracking position error vs. template update interval for a target

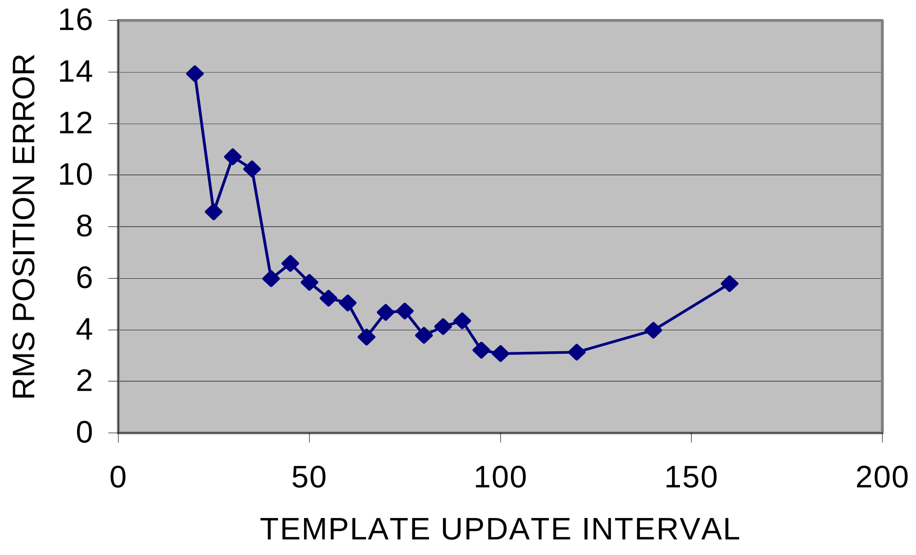


Figure 17. Tracking position error vs. template update interval for another target

Both experiments used the same set of images, straightforward motion with a 16 mm lens, but different features. The results indicate that different features may have somewhat different optimal values for template update interval. Too frequent update of the template image window causes the target to drift more and increases the rms position error, while too infrequent update also increases the position error and makes the tracker more likely to lose the target. An update interval of every 50 images (Figure 16) to every 100 images (Figure 17), which correspond to 0.5 m to 1 m distance change, respectively, appears to be a good value to use for straightforward motion. For roll and yaw motion, a good threshold value appears to be around 5º to 10º change to trigger the template update, although no extensive experiments were performed yet.

Although the current version of the 2-D tracker only allows the periodic template update of every n-th image or a fixed distance, it appears to be a better option to use the distance percent change to trigger template update, considering the image size change by perspective image formation. The orientation change is another parameter to monitor for template update. The ssd value or rms intensity error of the affine tracker is also a good parameter to monitor for template update. The affine tracker ssd would remain relatively flat until the tracker started to lose the target and then it would spike. The tracker would cache the last few images seen. When the affine tracker ssd crosses a threshold, the template for that target would be updated using the one with the lowest ssd value among the saved images. The template window would be updated by whichever occurs first: distance percent change, orientation change, or affine tracker ssd. Actual experiments of the above or similar ideas were not performed yet, because the 2-D tracker without active camera control prevented us from performing full 10-m target approach experiments. A more concrete experimental analysis on template window update must be deferred to the next 2-D/3-D tracking experiments with active camera control.

4.5 Maximum Image Displacement for Active Camera Control

Another goal was to determine the accuracy requirements for active camera control. If the feature tracker can track large image displacements, then the active camera control subsystem does not need to be as precise. To measure the capability of the feature tracker for tracking large image displacements, we first ran the tracker with no image skip on a set of targets that all tracked successfully. This provided a baseline of where each feature was located in each image which we used as a proxy for ground truth. We then increased image skip until features were lost. By comparing the runs where the features were lost to the baseline run where no features were lost, we evaluated exactly when a feature was lost and what image displacement occurred at that time. We also calculated the image displacement for all instances where features successfully tracked. The tracking instances were binned by image displacement, and within each bin the fraction of instances that tracked successfully over the total number of both successful and unsuccessful instances was calculated. This provides a statistical estimate of the probability of tracking a particular image displacement. This data is shown in Figures 17 and 18. The 15×15 window with 3 pyramid levels (Figure 18) tracked perfectly up to 12 pixels image displacement and quite well up to 30 pixels, while the 15×15 window with 2 pyramid levels (Figure 18) tracked perfectly up to 9 pixels and quite well up to 27 pixels. As a very rough approximation, if we assume the maximum trackable image displacement Δd_{max} is about the half the effective window size, we can compute it by

$$\Delta d_{max} \approx 1/2 * W * 2^{L-1},$$

where W is the window size and L is the number of pyramid levels. For window size 15, $\Delta d_{max} \approx$ 30 pixels with 3 pyramid levels, and $\Delta d_{max} \approx$ 15 pixels with 2 pyramid levels. Vertical dashed lines in Figures 18 and 19 mark these values. The approximation was good in Figure 18, but not obvious in Figure 19. In general, we observed that higher number of pyramid levels could handle larger image displacements. Good examples can be found in Figures 45 and 46 of Section 4.13, where 4 pyramid levels with 29×29 window yielded good tracking performance while 3 levels did not.

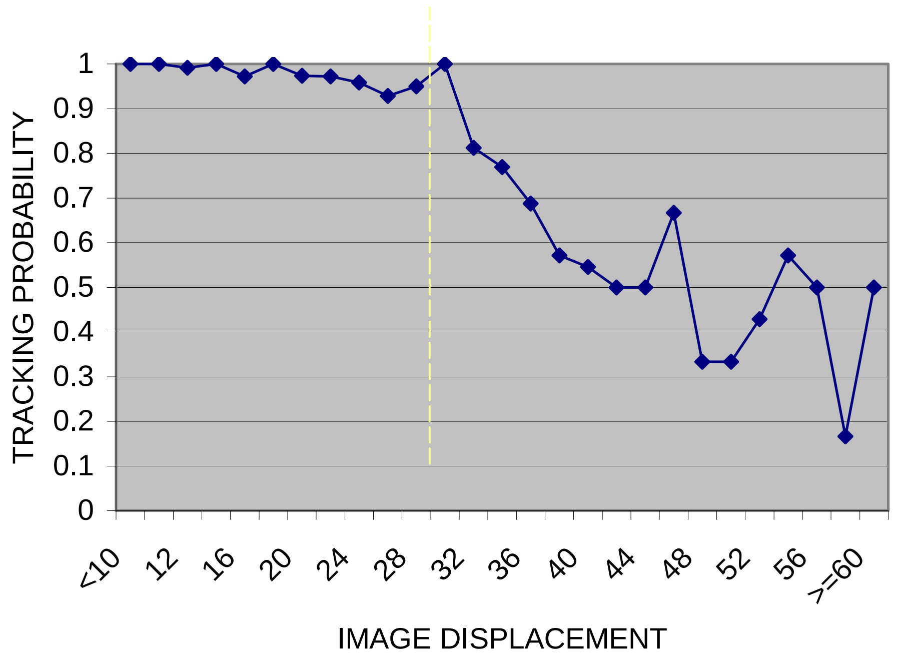


Figure 18. Tracking probability vs. image displacement for window size 15, pyramid level 3

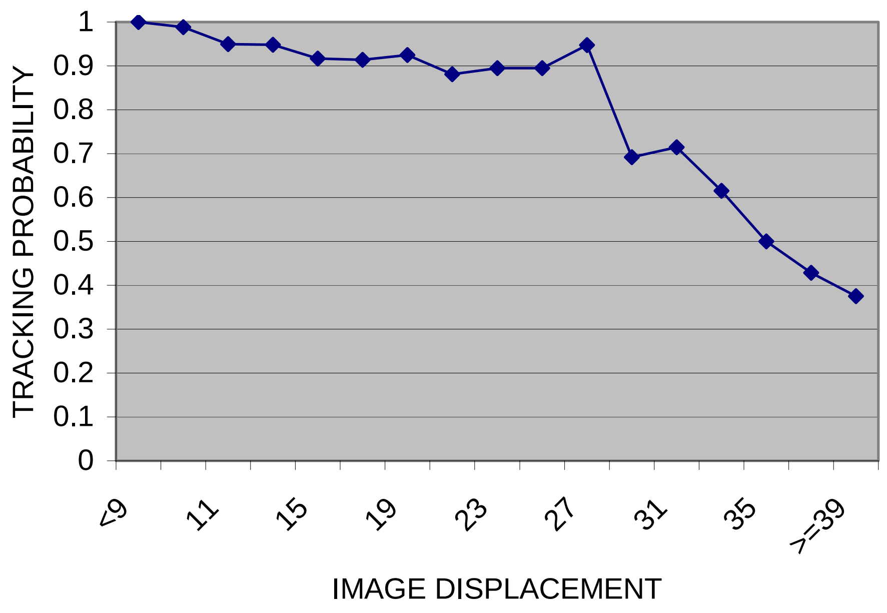


Figure 19. Tracking probability vs. image displacement for window size 15 pyramid level 2

4.6 Lighting changes

The effect of lighting was studied by performing 2-D tracking on the four sets of straightforward image sequences collected in four different time of the day: 9 AM, 2 PM, 4 PM, and 5 PM. For the first three sets, it was bright, sunny with no clouds. For the 5 pm set, it was still sunny with almost no clouds, but a patch of dark opaque clouds happened to pass across the sun during the image collection. Each set has different sunlight direction, forming different shadows. 2-D tracking performances of the four sets are summarized in Table 6. Images at the end of 3-m tracking were compared in Figure 20 for 9 am, 2 pm, and 4 pm (5-pm images were shown separately in Figure 21). Left three images are from 16 mm lens, while right three images are from 8 mm lens. The 2-pm images (middle pair) did not show much shadow over the rock surfaces, and resulted in 100% tracking. The 4- pm images made shadow in the left-hand side, behind rocks, but did not show much shadow on the rock surfaces facing the camera. They resulted in almost 100% tracking, where only one target for 8 mm drifted away about 10 pixels. The 9-am images made dark shadow on the rock surfaces facing the camera. At first glance, it appeared that the tracking performance would be very poor because of dark shadow. It turned out that surface images of rocks still had enough texture to track reasonably well even with dark shadow, resulting in 80% and 100% tracking performances. Only two targets for 8 mm lost tracking. Even though the lighting and shadow condition for 9 am was significantly poorer than those of 2 pm and 4 pm, the tracking performance was pretty good, demonstrating robustness of tracking relative to the sunlight direction change.

Image collection start time	F/L: 8mm	F/L: 4 mm
9 am; sun was in front, no cloud	100% (18 of 18 targets)	80% (8 of 10 targets)
2 pm; sun was above, no cloud	100% (18 of 18 targets)	100% (10 of 10 targets)
4 pm; sun was behind, no cloud	94% (17 of 18 targets)	100% (10 of 10 targets)
5 pm; a patch of cloud crossing sun	44% (8 of 18 targets)	0% (0 of 10 targets)

Table 6. Tracking performance at three different sunlight directions (9 am, 2 pm, 4 pm) and at one abrupt sunlight change condition (5 pm)

Tracking performance at 5 pm (Table 6) with dramatic sunlight changes was very poor. For the 8 mm lens, 8 out of 18 initial targets were suddenly lost after 0.75 m tracking, and 2 more targets were lost later, resulting in 44% tracking over 3 m. For the 4 mm lens, 7 out of 10 initial targets were suddenly lost after 0.75 m tracking, and all targets were lost by 1.75 m, resulting in 0% tracking over 3 m. Images in Figure 21 clearly explain why many targets were suddenly lost. The left three images were from 8 mm lens, and the right three were from 4 mm lens. Three distinct images at near 0.75 m of tracking were shown in the order of time from top to bottom, when a patch of dark, opaque cloud started to pass across the sun. Right before the cloud covered the sun, the shadows were stark (top pairs in Figure 21). When a patch of opaque cloud partially covered the sun, the surface of the Mars Yard became darker, and shadows became dimmer (middle pair in Figure 21). When the sun was fully covered, shadows completely disappeared (bottom pair). In order to compensate for lighting change, automatic gain control (AGC) was activated for both 8-mm and 4-mm images during our data collection. The automatic gain control tried to keep the average intensity relatively constant over the entire image regardless of ambient light changes. In Figure 21, the 8-mm images did not include the “bright” sky in view. For 8-mm images (left three images in Figure 21), automatic gain control was effective to maintain the average brightness of the Mars Yard at about the same level, even with the dramatic reduction in direct sunlight (bottom left image in Figure 21). By contrast, the 4-mm images included the “bright” sky in view.

The sky was still very bright because there were no other patches of clouds but near the sun. For 4-mm images, automatic gain control failed to effectively compensate for ambient lighting changes on rock surfaces because of the “bright” sky in view. Note that Mars Yard surface got much darker (bottom right image in Figure 20). The effectiveness of AGC was the reason why tracking performance with 8 mm was significantly better than that of 4 mm during the dramatic sunlight changes. The AGC, however, did not avoid shadow contrast changes in images from stark shadow to almost no shadow, which could cause 2-D tracking less reliable.

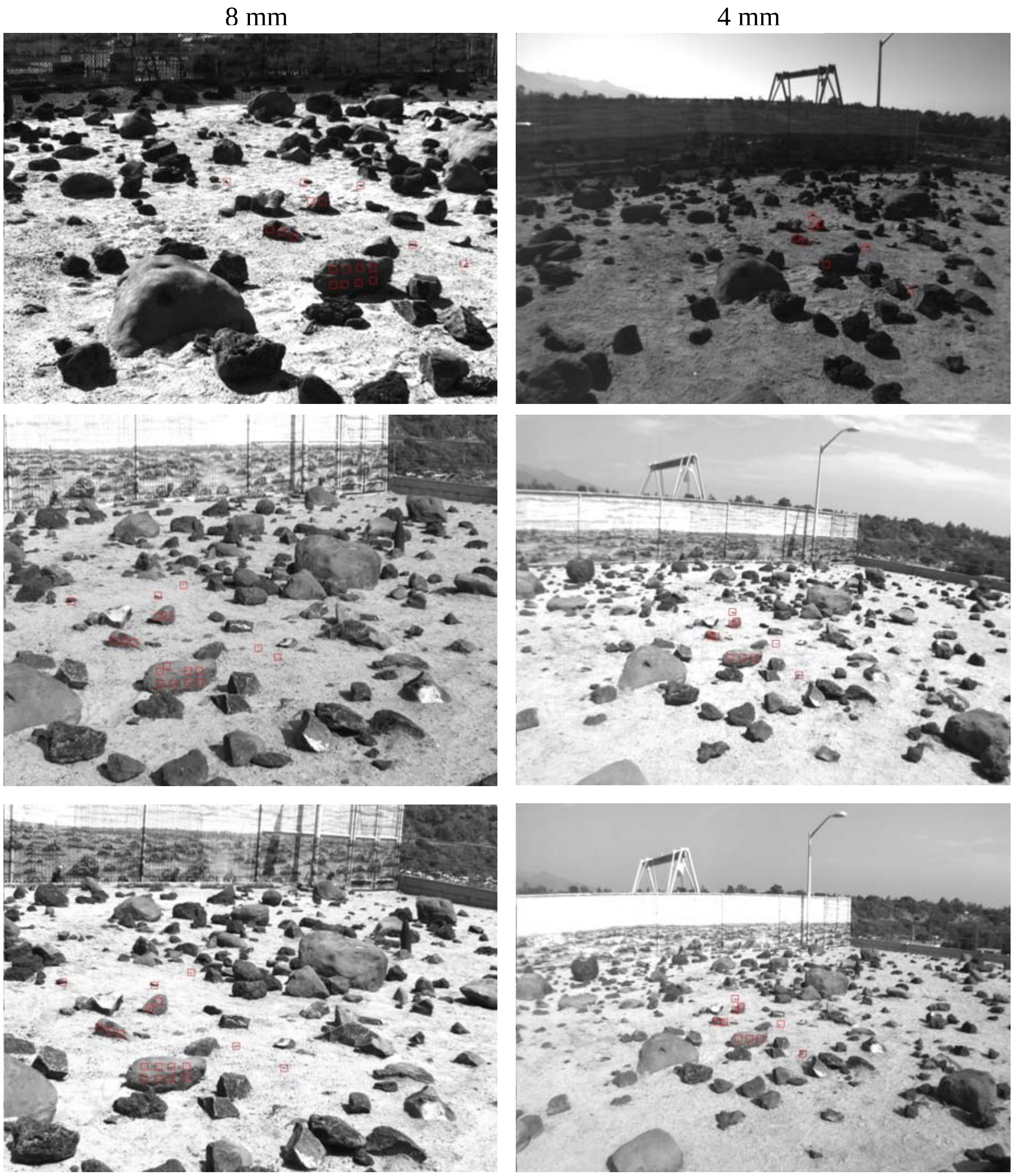


Figure 20. Images at the end of 3-m forward tracking for 8 mm lens (left) and 4 mm lens (right): at 9 AM (top pair), at 2 PM (middle pair), and at 4 PM (bottom pair)

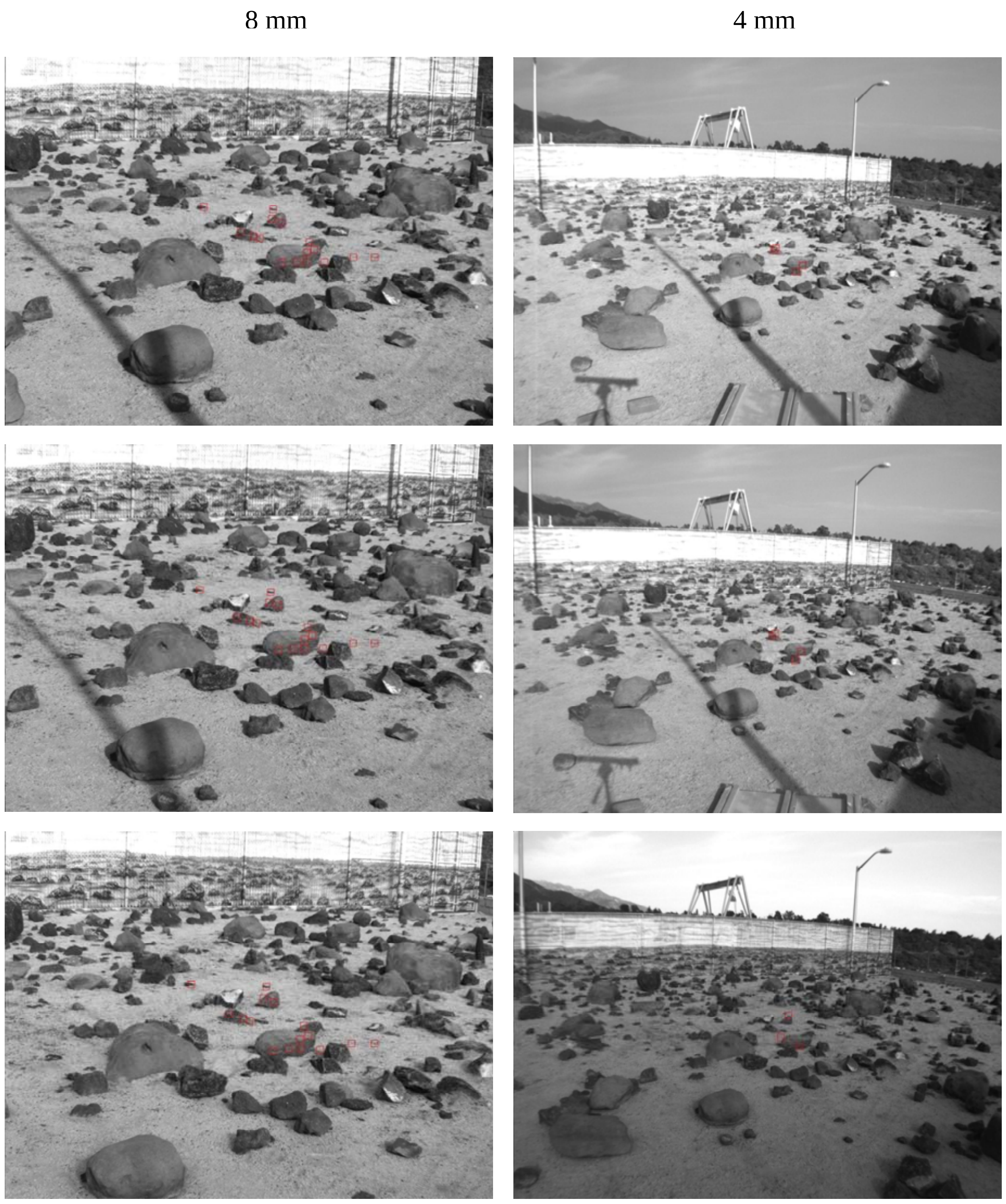


Figure 21. Images at the end of 3-m forward tracking for 8 mm lens (left) and 4 mm lens (right): with full sun (top pair), with sun partially covered (middle pair), and with sun fully covered by a patch of clouds (bottom pair).

Incidentally, this dramatic lighting change due to dark clouds (opacity > 10) would not happen in Martian environment. Here is a direct email quote from Leslie Tamppari, who was Deputy Project Scientist for MSL, regarding Martian clouds and sun lighting [Tamppari, 2003].

“The clouds on Mars are very thin clouds - more like stratospheric clouds on earth. The typical visible opacity is only around 0.4 or so. They appear to form in haze layers most of the time, but there can be lee waves associated with topographic rises, but again the clouds are thin for the most part. Occasionally they are thicker, but usually associated with mountaintops and we won't be going there. Dust will also occur on Mars and could reach opacities higher than the clouds. In dust storm conditions, the opacities can get above 2. However, these are likely to be hazy as well.

Sunlight conditions will vary as a function of time of year and latitude (as well as time of day, of course). If we are at far N or S latitudes during winter, we could have the sun low in the horizon (5 degrees above for a few months).”

4.7 Tracking with Oblique and Sideways Camera Motion

2-D tracking was performed for the image sequences collected with oblique and sideways camera motions. The camera was moved on a liner motion stage, so that consecutive images did not show large image displacements. Since the camera orientation was not changed during the image collection, oblique and sideways camera motions tended to lose targets out of the camera view more quickly than forward camera motion. Oblique-motion allowed 2-D tracking up to 1.6 meters with a 16 mm lens (Figure 22) and over the entire 4 meters with 4 mm lens (Figure 23). Sideways motion allowed tracking up to 1.2 meters with a 16mm lens (Figure 24) and up to 3.3 meters with an 4 mm lens (Figure 25). Oblique and sideways camera motions are similar to straightforward camera motions in terms of tracking, because they all cause translation and scale changes in images. Since oblique and sideways camera motions have the problem of targets going out of sight quickly, we decided not to spend further time evaluating oblique and sideways camera motions. Instead, we focused on examining 2-D tracking with straightforward camera motions for translation and scale changes in images.

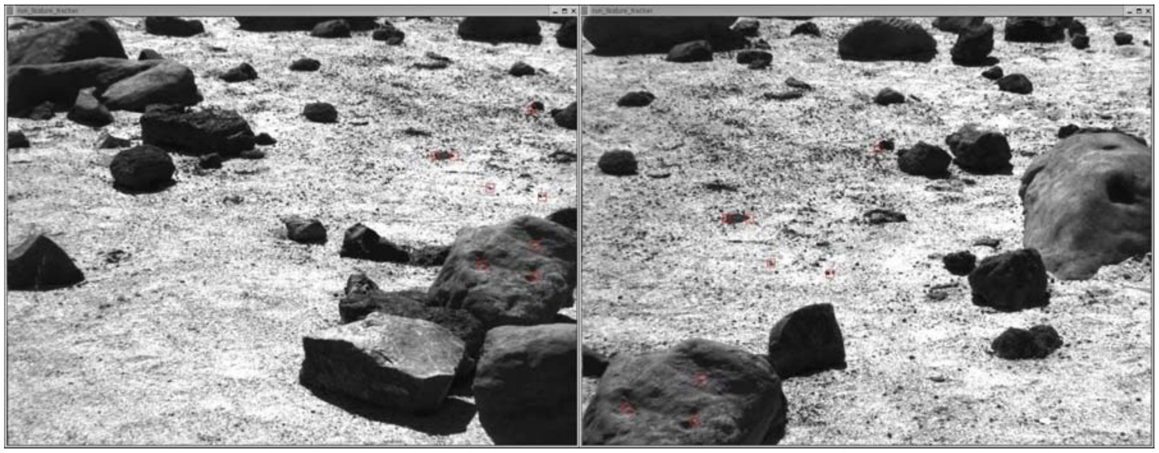


Figure 22. 2-D target tracking with oblique camera motion using a 16 mm lens: initial image (left) and end image after 1.6 m oblique move (right).

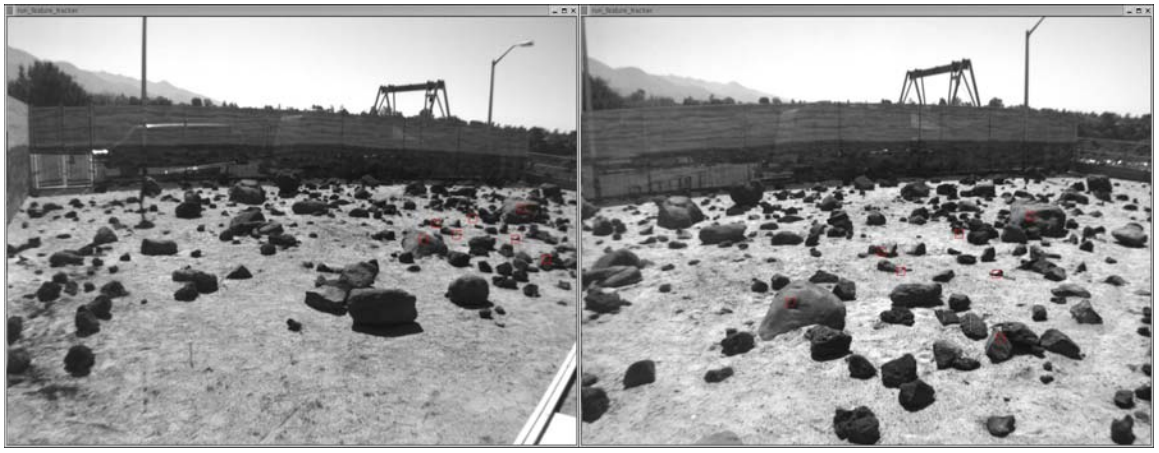


Figure 23. 2-D target tracking with oblique camera motion using a 4 mm lens: initial image (left) and end image after full 4-m oblique move (right).

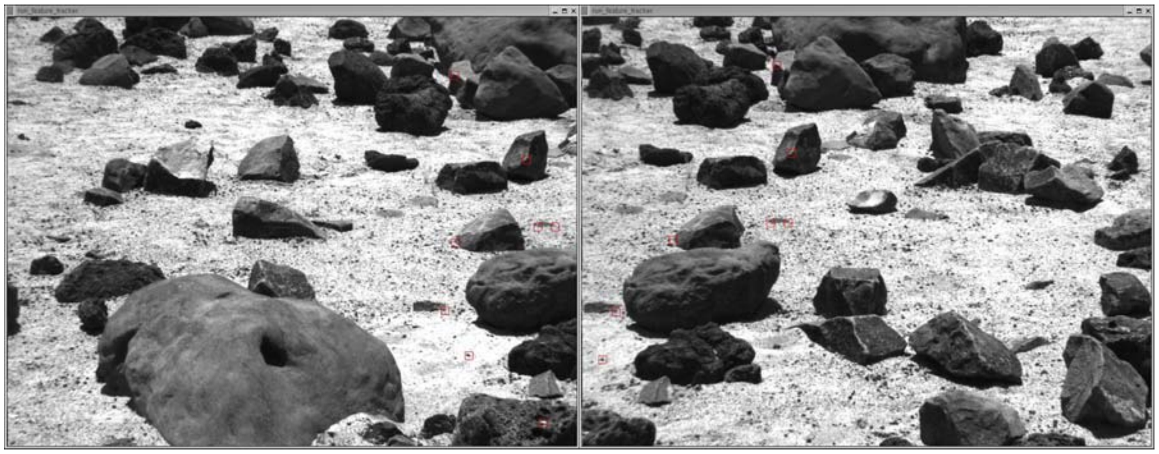


Figure 24. 2-D target tracking with sideways camera motion using a 16 mm lens: initial image (left) and the end image after 1.2 m sideways move (right).

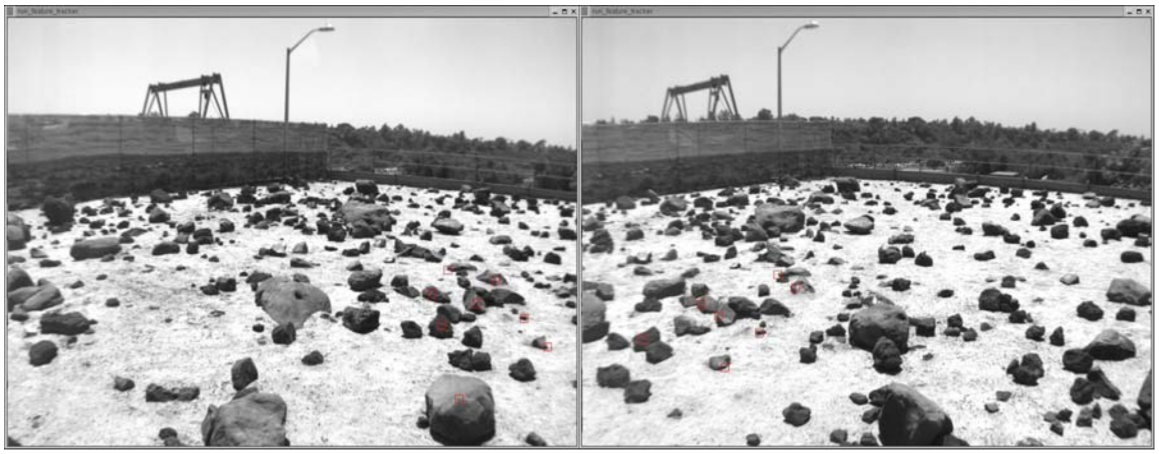


Figure 25. 2-D target tracking with sideways camera motion using a 4 mm lens: initial image (left) and the end image after 3.3 m sideways move (right).

4.8 Tracking with Actual Rocky8 Mast Camera Images

Figure 26 shows a series of the Rocky8 live image data. Rocky8 mast camera images were collected every 3 cm over a 10 m straightforward rover motion. Targets tracked well for the first 1.8 m (top right of Figure 26) until the rover went over the rock, which caused large image displacements (sometimes over 100 pixels). By 2.4 m, all targets were lost. Again, it clearly indicates that active camera control by 2-D/3-D tracking is essential. Since we will be evaluating 2-D/3-D tracking next, we decided not to spend further time evaluating actual Rocky8 mast camera images without active camera control.

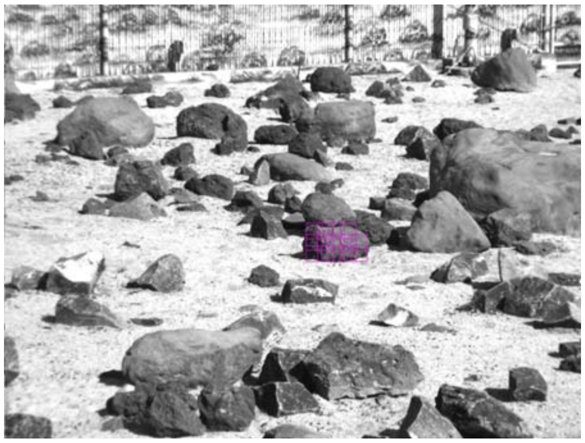
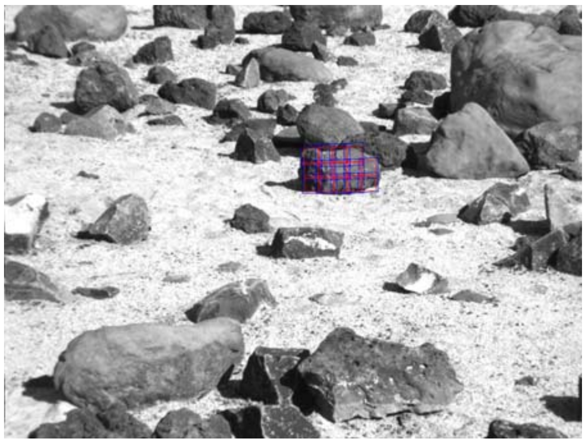
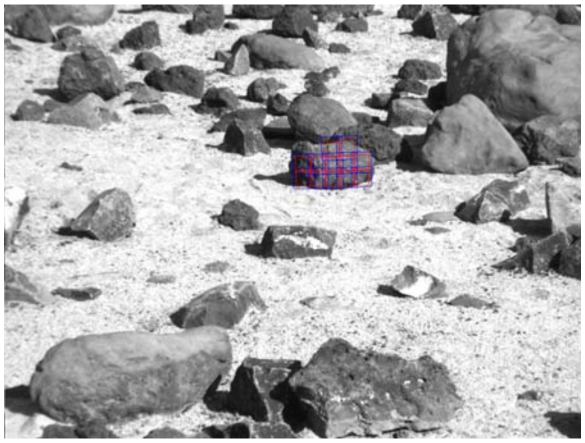
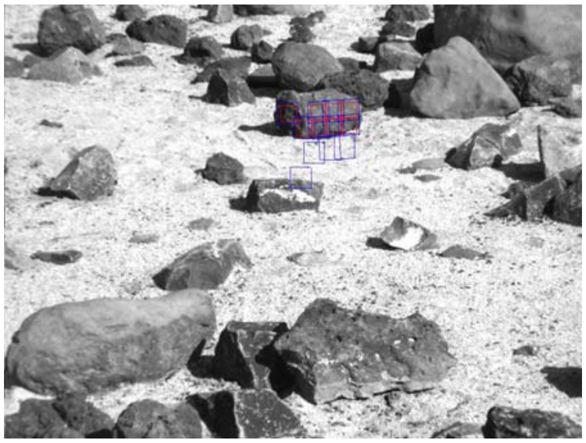
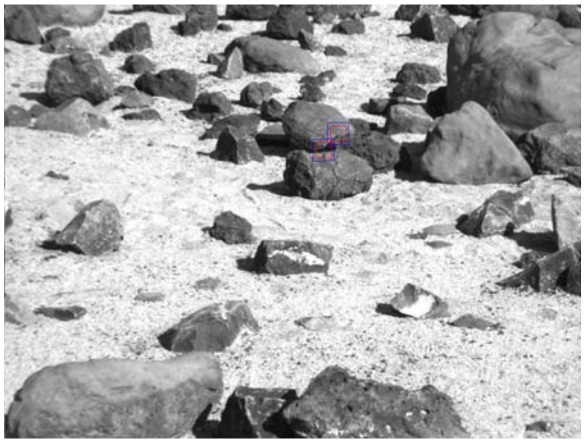






Figure 26. 2-D tracking with actual Rocky8 mast camera images: at initial position (top left), after 1.8 m (top right), after 2.1 m (middle left), 2.25 m (middle right), and 2.4 m (bottom left) forward motion.

4.9 Average Tracking Performances from Forward, Roll, and Yaw Camera Motions

To collect a statistical overview of 2-D tracking performance, we ran 2-D tracking on three image sequences of 4-m straightforward, 90° roll, and 45° yaw camera motions. Tracking targets were selected over several rocks of small, medium, and large image sizes. Tracking parameters were selected to be values that in our experience worked well in a wide variety of situations as described in previous Sections. The feature tracking window size was set to 15 pixels × 15 pixels in all tests. The number of pyramid levels was set to 3 for straightforward camera motion, and set to 4 for roll and yaw camera motions to accommodate larger image displacements. The template update interval was set to 10 or every 10-th image processed. This corresponds to every 50 cm for forward motion (with imag skip of 4), every 10° for roll moton (no image skip), and every 2.5° for yaw camera motion (no image skip). Table 7 shows the test results of 2-D tracking for forward, roll, and yaw camera motions with 16, 8, 4, and 2.3 mm lenses. The average tracking performances were 80% to 100%.

	16 mm (17°x13°)	8 mm (33°x25°)	4 mm (65°x49°)	2.3 mm (113°x86°)
Forward motion	65/65 (100%)	40/45 (89%)	31/31 (100%)	23/23 (100%)
Roll motion	13/16 (81%)	14/14 (100%)		
Yaw motion	10/12 (83%)	14/16 (88%)		

Table 7. Average tracking performance with forward, roll, and yaw camera motions.

Figures 27-30 show beginning and end images of 2-D tracking with forward camera motions for 16, 8, 4, and 2.3 mm, respectively. For the 16 mm lens (Figure 27), all selected targets including the ones on large-image-size rocks tracked well with a 100% success rate. Each target seems to have sufficient texture to track. For the 8 mm lens (Figure 28), targets on small-image-size and medium-image-size rocks tracked 100%, while 5 out of 25 targets failed. The failed targets are from a large-image-size rock, and there seems to be not enough texture to track well. For the 4 mm lens (Figure 29) and 2.3 mm lens (Figure 30), all targets tracked well with 100% success.

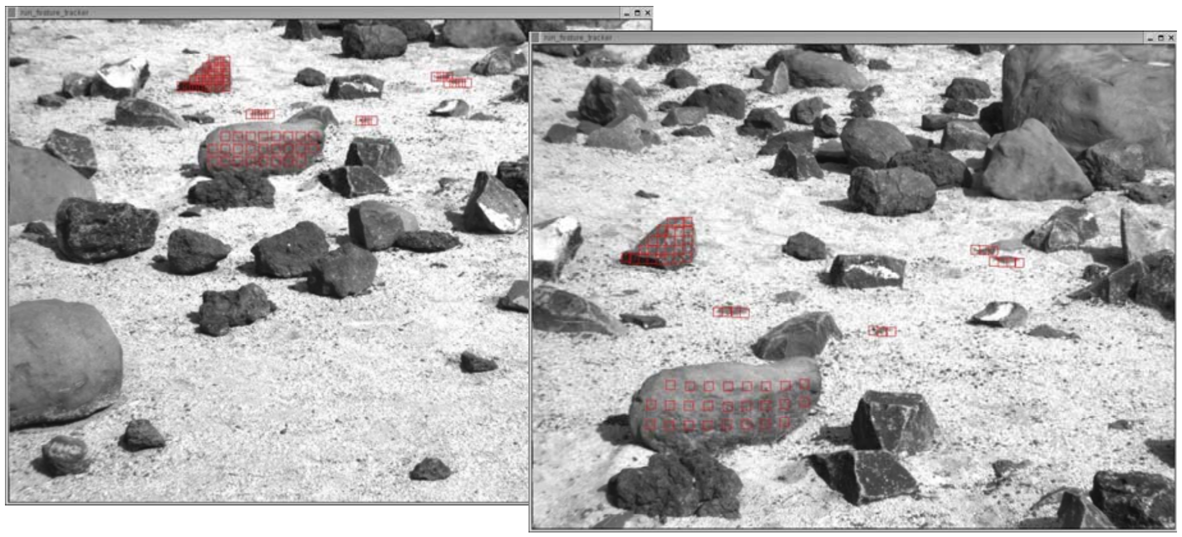


Figure 27. Beginning (left) and end (right) images of 4-m straightforward camera motion with 16 mm lens. Targets were selected on several rocks of small, medium, and large image sizes.

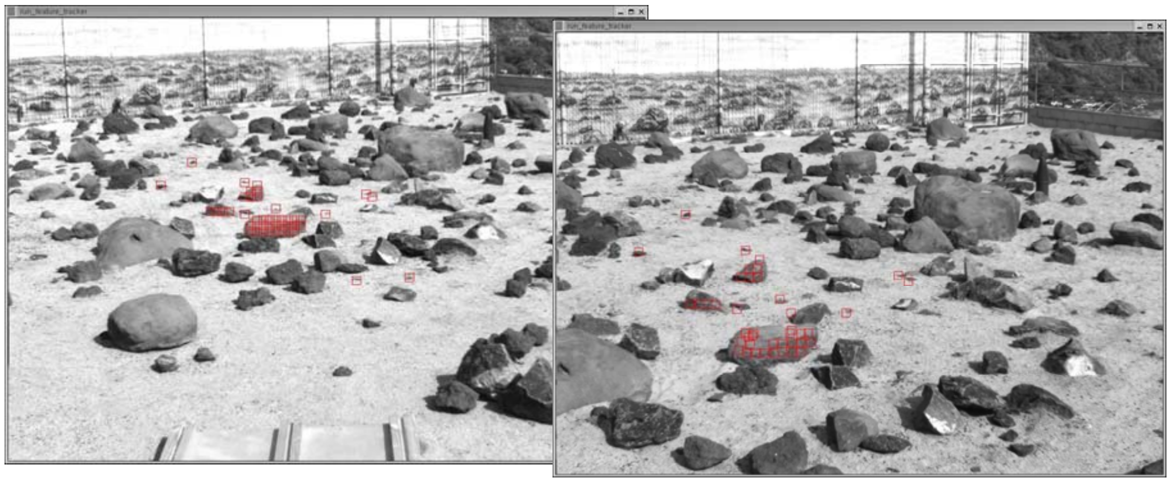


Figure 28. Beginning (left) and end (right) images for 4-m forward motion with 8 mm lens

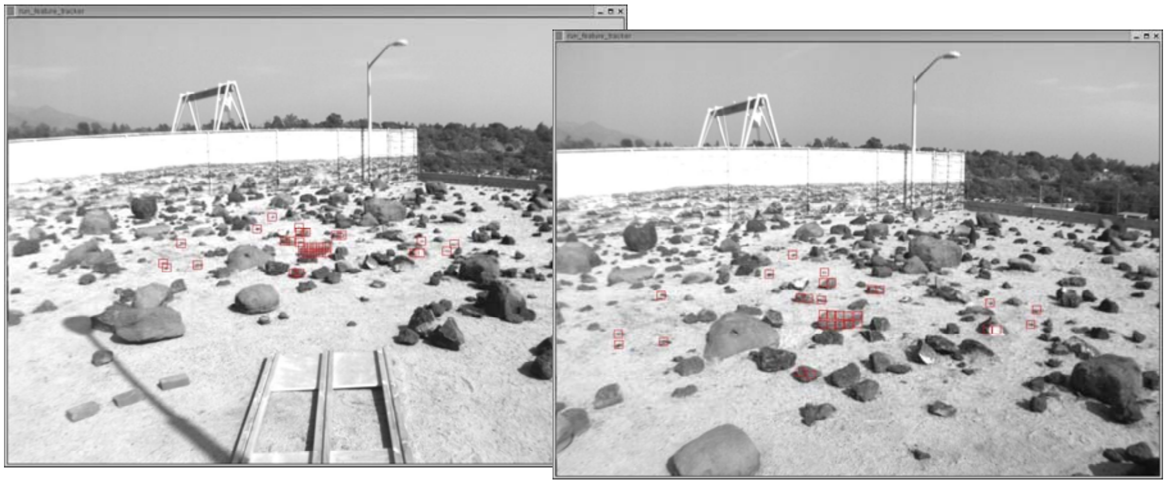


Figure 29. Beginning (left) and end (right) images for 4-m forward motion with 4 mm lens

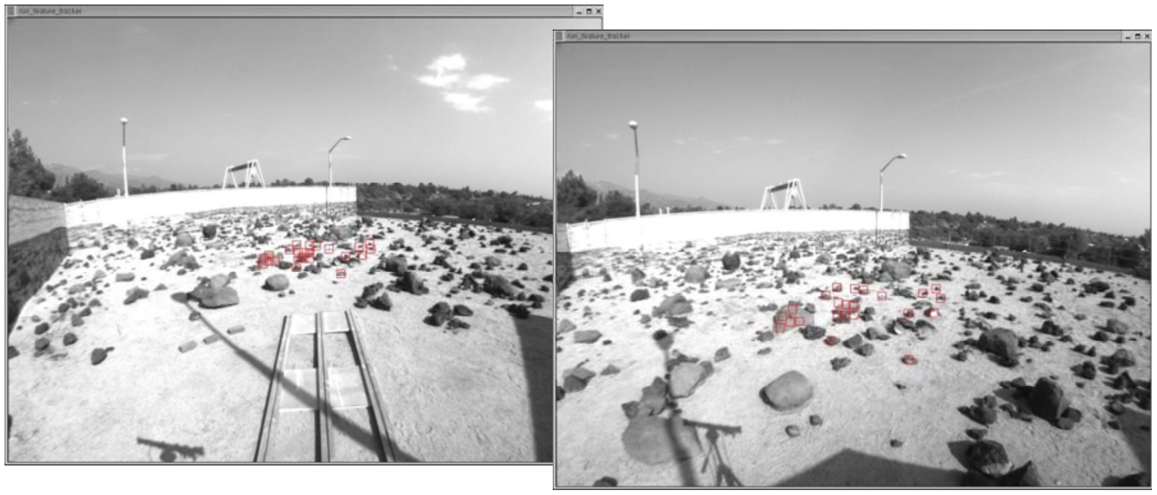


Figure 30. Beginning (left) and end (right) images for 4-m forward motion with a 2.3 mm lens

Figures 31 and 32 show beginning and end images of 2-D tracking with roll camera motions for 16 and 8 mm, respectively. Three targets were lost out of 16 selected targets for the 16 mm lens (Figure 31), while all targets were tracked for the 8 mm lens (Figure 32).

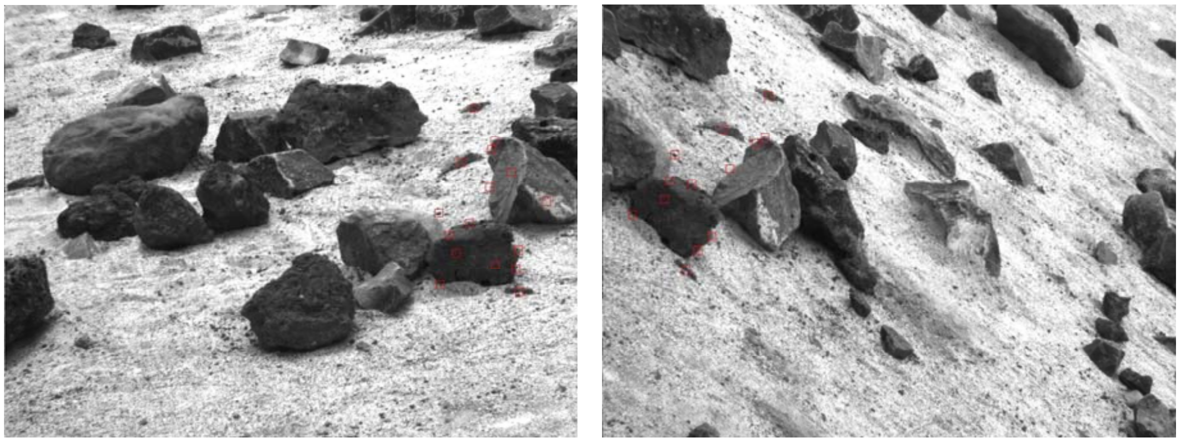


Figure 31. Beginning (left) and end (right) images for 40º roll motion with 16 mm lens. Targets were out of camera view beyond the 40º roll motion selected.

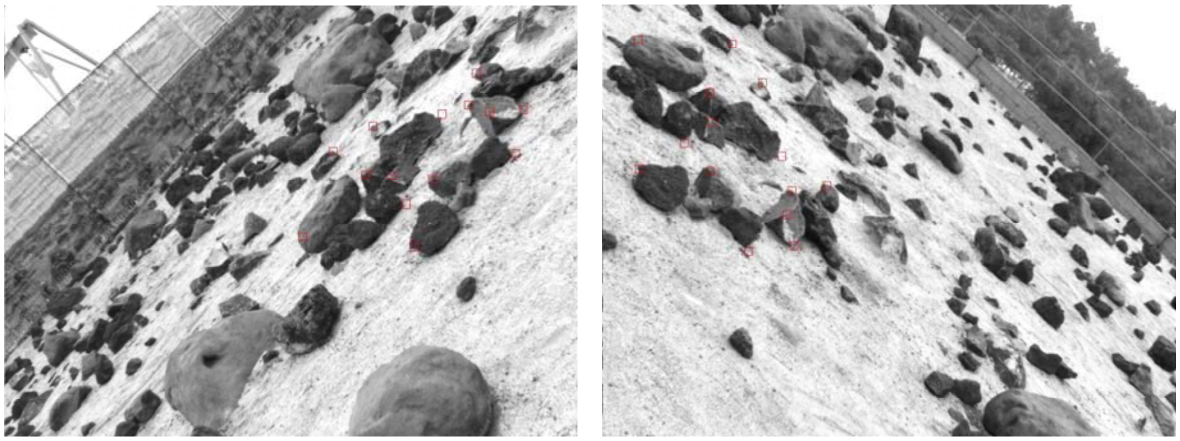


Figure 32. Beginning (left) and end (right) images for 90º roll motion with 8 mm lens.

Figures 33 and 34 show beginning and end images of 2-D tracking with yaw camera motions for 16 and 8 mm, respectively. One target was lost out of 12 selected targets for the 16 mm lens (Figure 33), while two targets were lost out of 16 for the 8 mm lens (Figure 34).

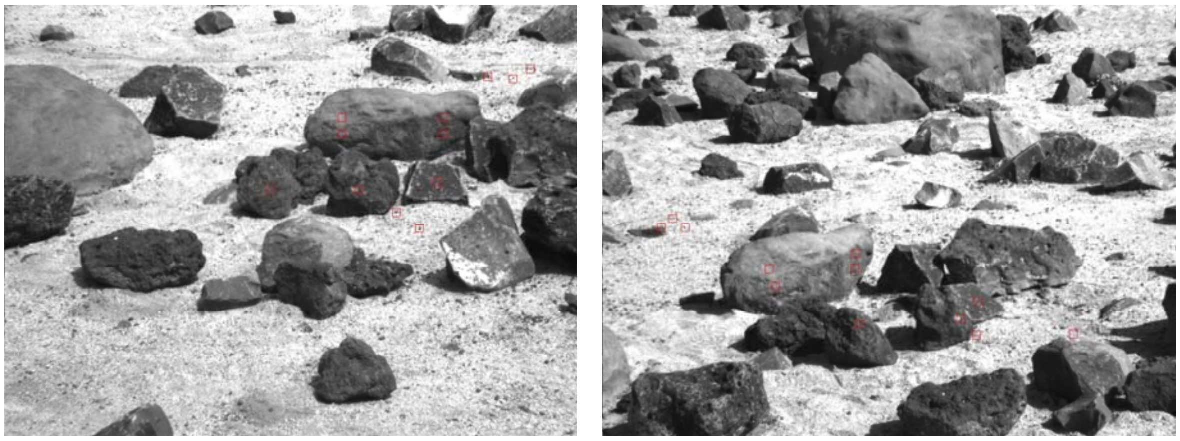


Figure 33. Beginning (left) and end (right) images for yaw camera motion with 16 mm lens

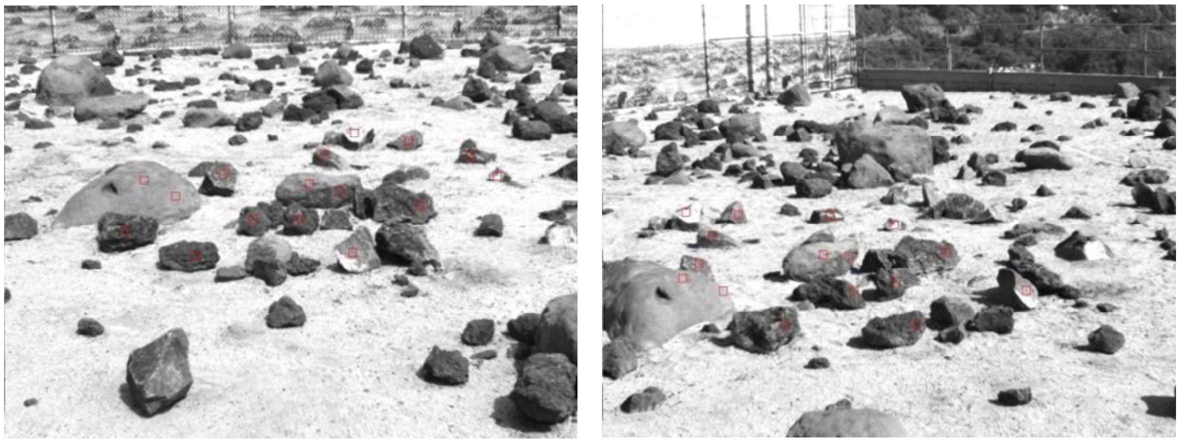


Figure 34. Beginning (left) and end (right) images for yaw camera motion with 8 mm lens

4.10 Tracking with Small-Image-Size Rocks

Since the average tracking performance was only 80% to 100% as described in Section 4.11, we investigated further as to when tracking fails and how it can be improved. More careful examination revealed that 2-D tracking with small-image-size rocks was 100% successful for all forward, roll, and yaw camera motions, as long as target rocks were not occluded during the course of tracking. Figure 35 shows such an example of tracking targets on several small-image-size rocks with roll camera motions. Red squares superimposed on the images represent updated tracking windows, which were always reset to the upright orientation for computational efficiency. Thus, four corners of tracking windows between images do not correspond to each other; only the center position does. A good way to observe how well a combined 2-D translation and affine transform tracking works is to display the affine transformed quadrilateral windows corresponding to the initial tracking square windows. In Figure 35, after tracking over 70° roll camera motion, all affine transformed quadrilateral windows (blue) are about 70° rotated relative to the upright feature square windows (red) without much scale change. This clearly depicts an excellent tracking performance.

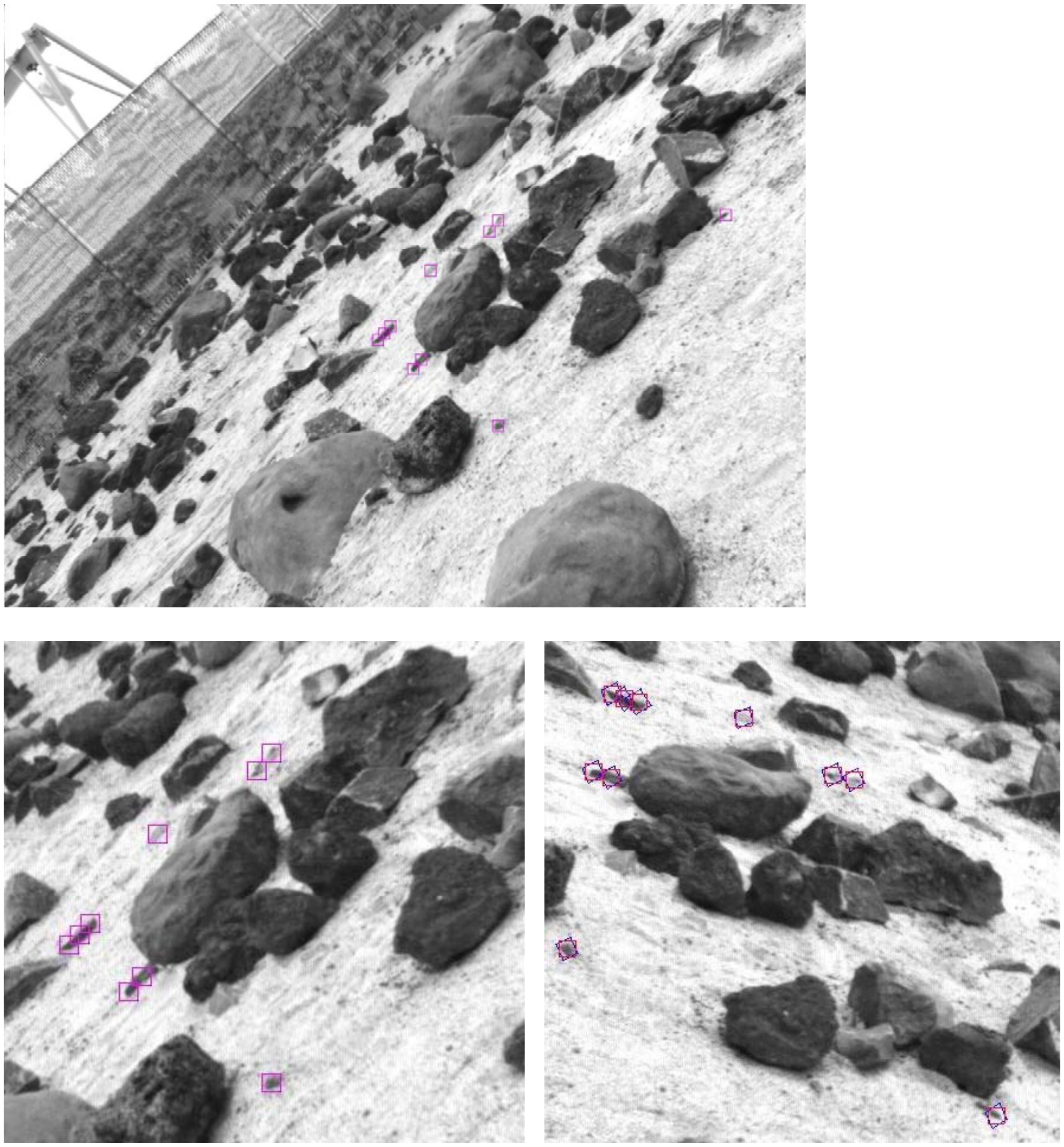


Figure 35. 2-D tracking of small-image-size rocks over 70º roll camera motion with 8 mm lens. A portion of the beginning image (top) was enlarged (bottom left), showing initial target feature windows in red squares. All targets tracked well using a 15×15 window with 3 pyramid levels. Some targets were about to move out of field of view of the camera at roll = 25° (bottom right). Affine transformed windows (blue quadrilaterals) show an excellent tracking performance.

4.11 Forward-Motion Tracking with Large-Image-Size Rocks

For forward camera motion, the biggest problem was to track targets selected on large image-size rocks with the 8mm lens. Unlike targets on small-image-size rocks, some of these targets lost tracking. Figure 36 show tracking results for 15×15 window with 4 pyramid levels. Only 7 out of 12 targets or (58%) tracked. Affine transformed windows for these lost or strayed targets were often greatly deformed. It appeared that the lost targets did not have sufficient texture within the tracking windows. So we approached this problem by increasing the feature tracking window size to include more texture.

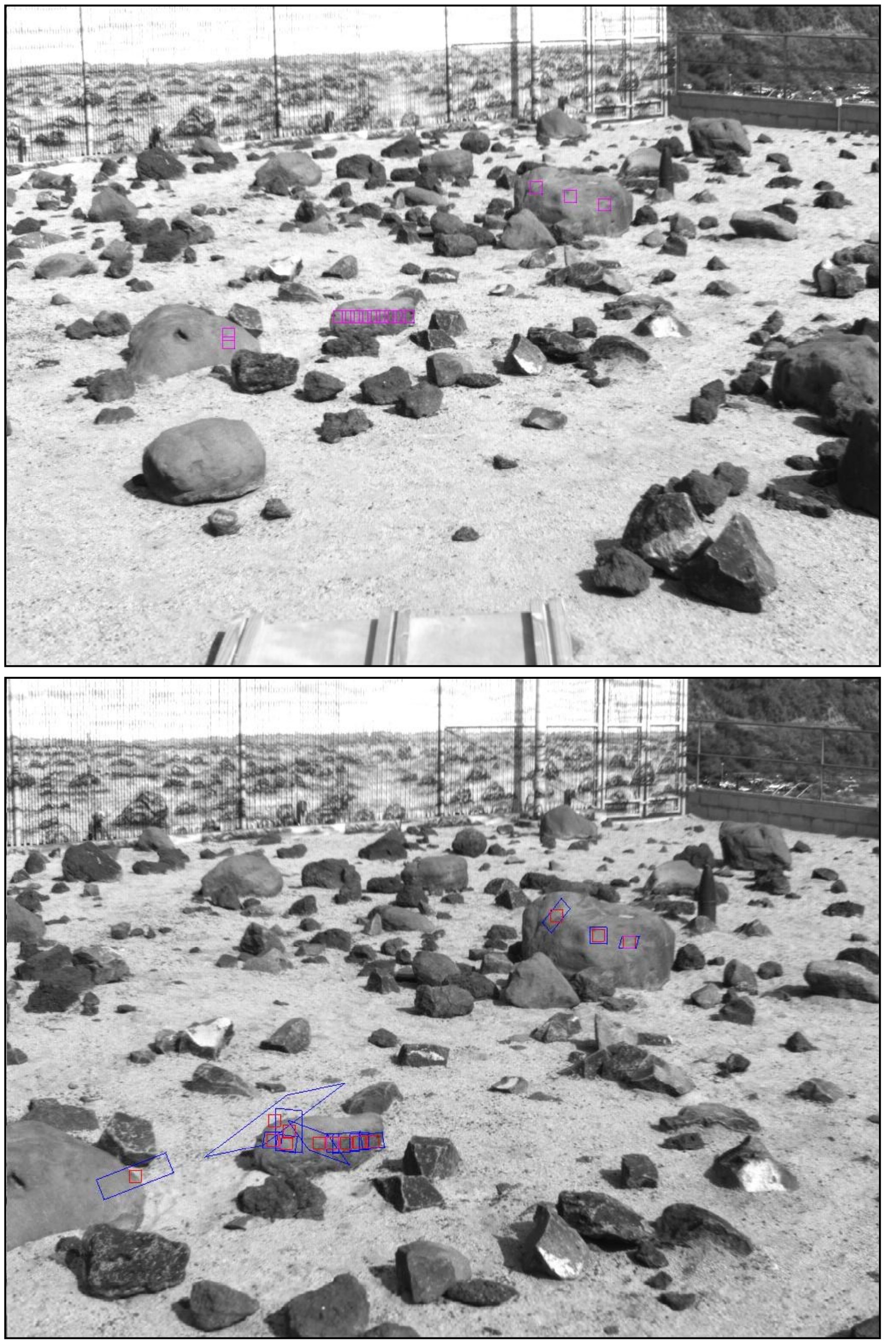
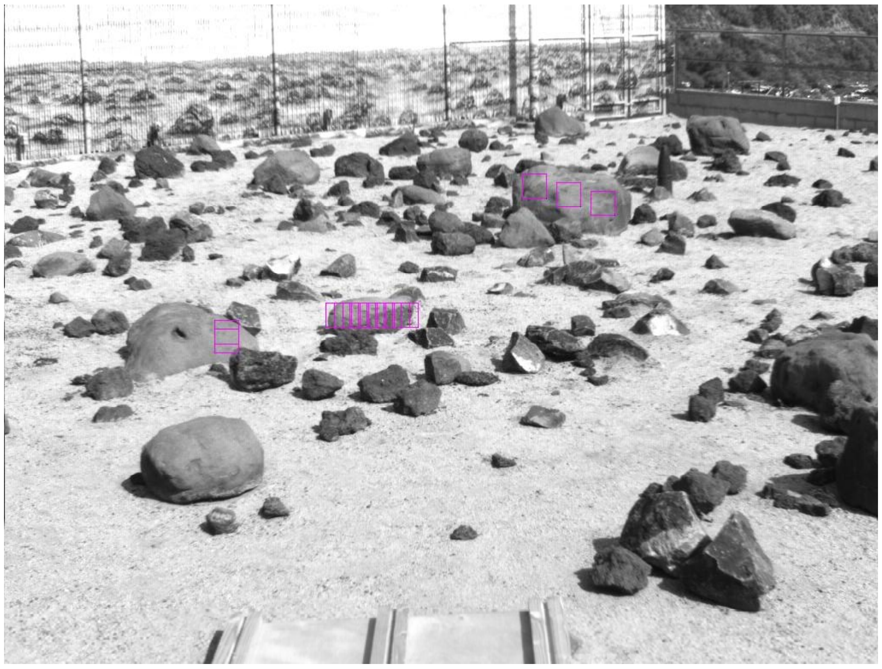


Figure 36. Initial image (top) and end image after tracking over 4-m forward motion (bottom) using 15×15 window with 4 pyramid levels.

When the window size was increased to 29×29 with 3 pyramid levels, 10 out of 12 targets tracked. Two lost targets did not have much feature to track, and drifted over the surface of the rock. The 6-parameter affine transform was a lot more sensitive to low-texture targets than the 2-parameter 2-D translation, resulting in affine transformed windows badly deformed.



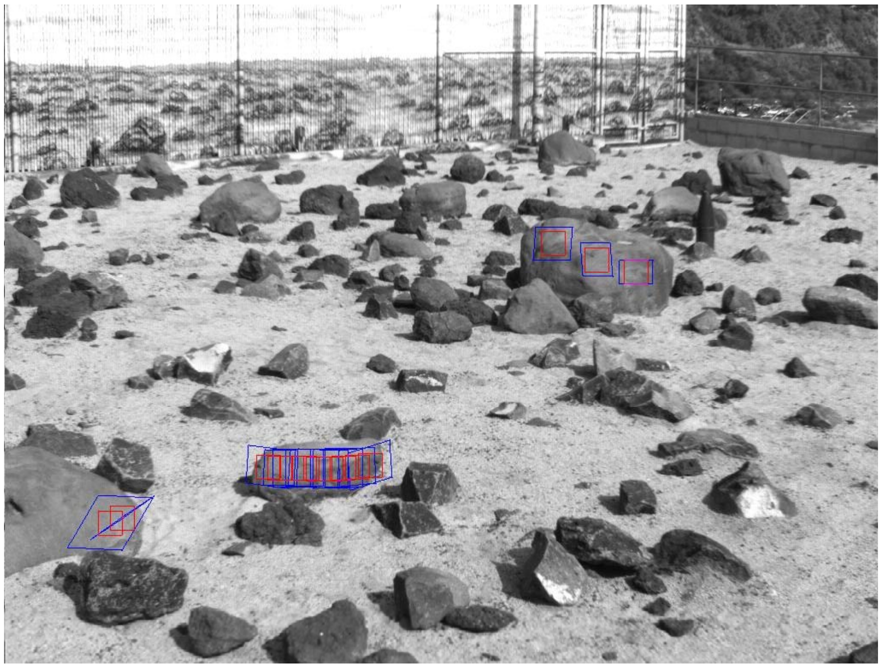


Figure 37. Initial image (top) and end image after tracking over 4-m forward motion (bottom) using 29×29 window with 3 pyramid levels.

When the feature track window size was further increased to 59×59 with 2 pyramid levels (Figures 38 and 39), all 12 targets tracked. Since the current software could not run all 12 targets in a single run, we ran in two separate runs. All 12 targets tracked well in terms of the center positions of the targets. We noted, however, that affine transformed windows were pulled towards adjacent occluding/occluded rocks.

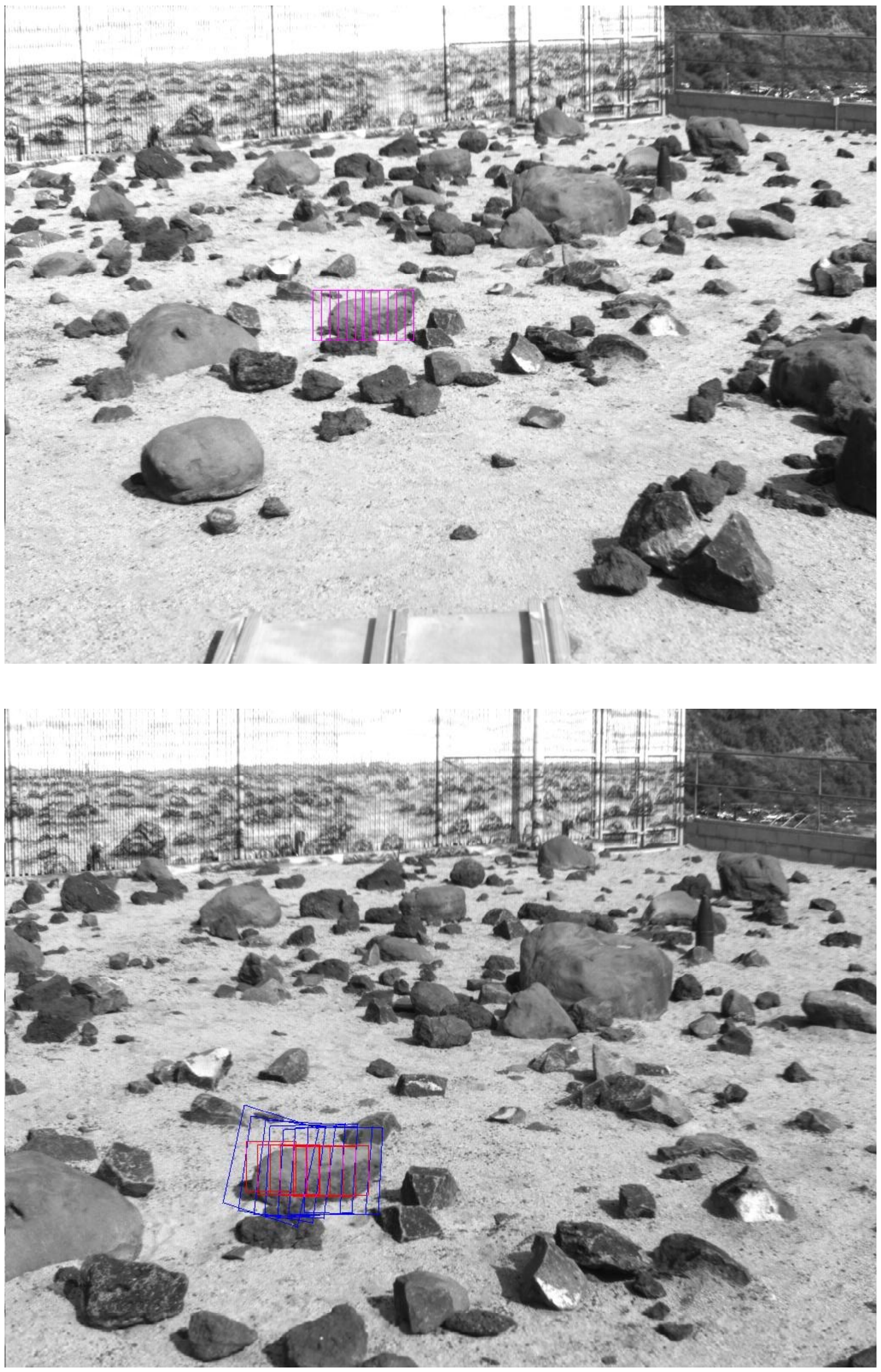


Figure 38. Initial image (top) and end image after tracking over 4-m forward motion (bottom) using 59×59 window with 2 pyramid levels. More targets were selected in Figure 39.

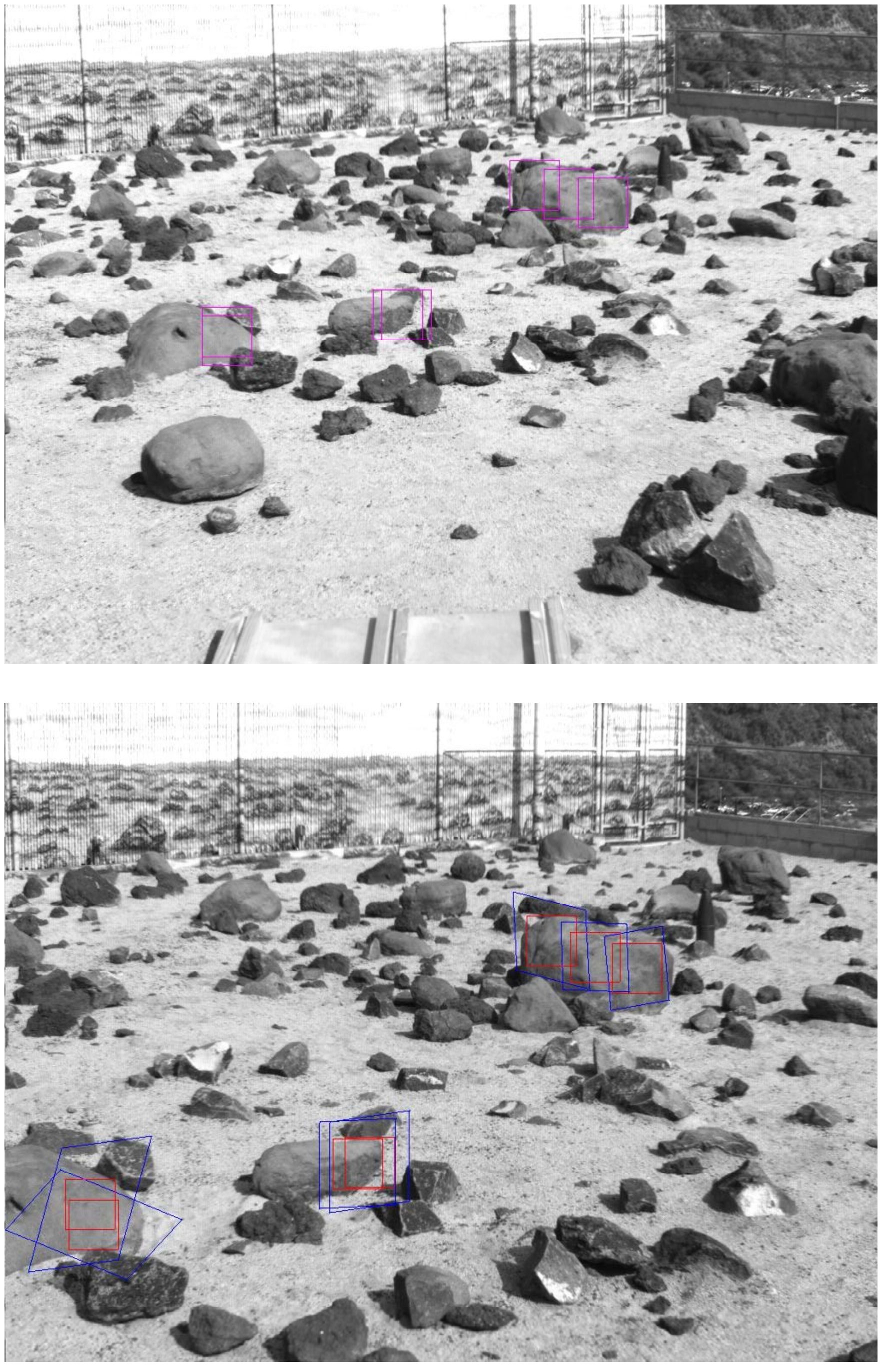


Figure 39. Initial image (top) and end image after tracking over 4-m forward motion (bottom) using 59×59 window with 2 pyramid levels. Different targets were selected here.

Two affine transformed quadrilaterals were oriented quite differently for the lower-left rock in the bottom image of Figure 39. One was pulled towards the occluding rock in the front, while the other was pulled towards the occluded rock behind. Several affine transformed quadrilaterals for both upper-right and lower-middle rocks were also pulled by the occluded distinct rocks behind. As the camera moves, relative rock locations in the image change. This causes the affine transform to be distorted inaccurately.

4.12 Roll-Motion Tracking with Large-Image-Size Rocks

For roll camera motions, sometimes targets selected from large-image-size rocks tracked 100% successfully even with the relatively small 15×15 window. For instance, in Figure 40 for an 8 mm lens, all 15 targets selected from a large rock tracked well over the entire 90º roll using 15×15 window size with 4 pyramid levels.

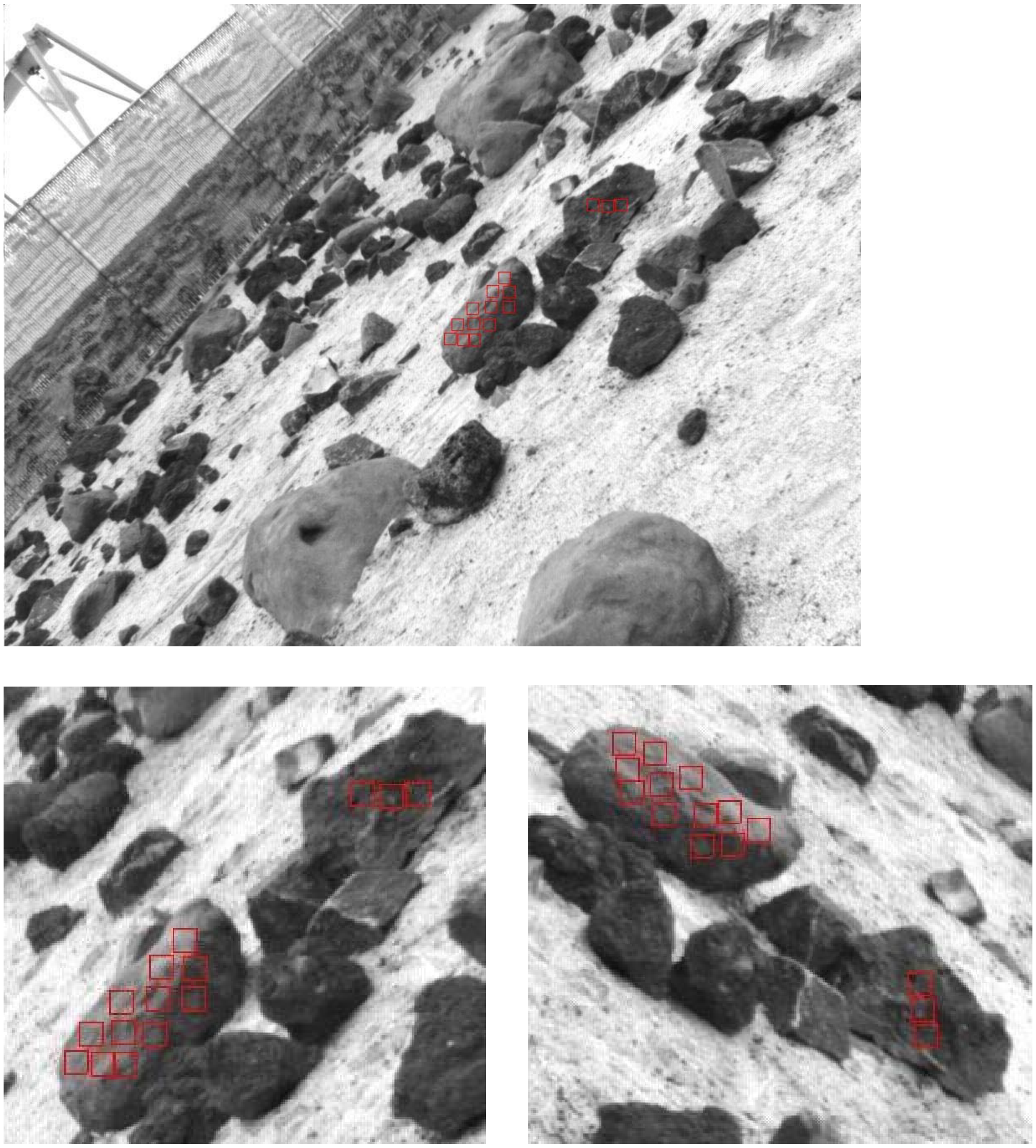


Figure 40. All 15 targets selected on two large-image-size rocks tracked well through a 90° roll camera motion, using a window size of 15×15 and 4 pyramid levels: initial image (top), its close-up (bottom left) showing targets selected, and an enlarged portion of the end image after the 90º roll (bottom right).

Another example of good tracking is shown in Figure 41 for 16 mm lens. All 5 targets tracked well through a 60° roll using 15×15 window with 3 pyramid levels.

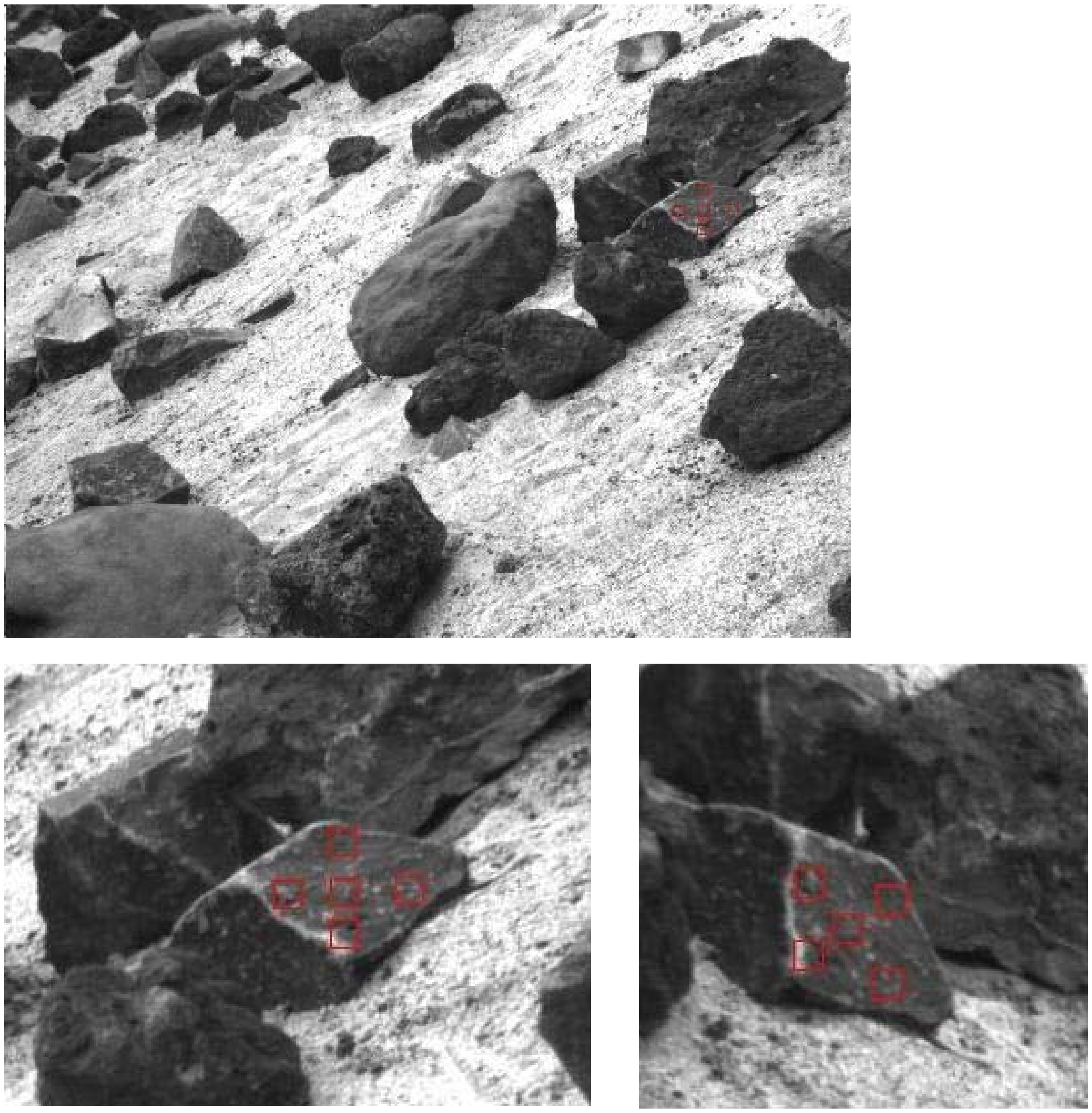


Figure 41. All 5 targets selected on the face of another large-image-size rock also tracked well using a window size of 15×15 and 4 pyramid levels: initial image (top), its close-up (bottom left) showing targets selected, and an enlarged portion of the end image after 90º roll (bottom right).

In the third example, however, 15×15 window size with 4 pyramid levels was not large enough to achieve 100% tracking. In Figure 42 for 16 mm lens, 1 target out of 12 on a large-image-size rock got lost over 40° roll change. Further, several of affine transformed windows were quite poor. The tracking was done up to 40°, because the rock went out of camera view afterwards. When the window size was increased to 29×29 with 3 pyramid levels (Figure 43), all 12 targets tracked very well. In particular, affine transformed windows were also very good. This clearly demonstrates that larger window helps tracking for low-texture targets.

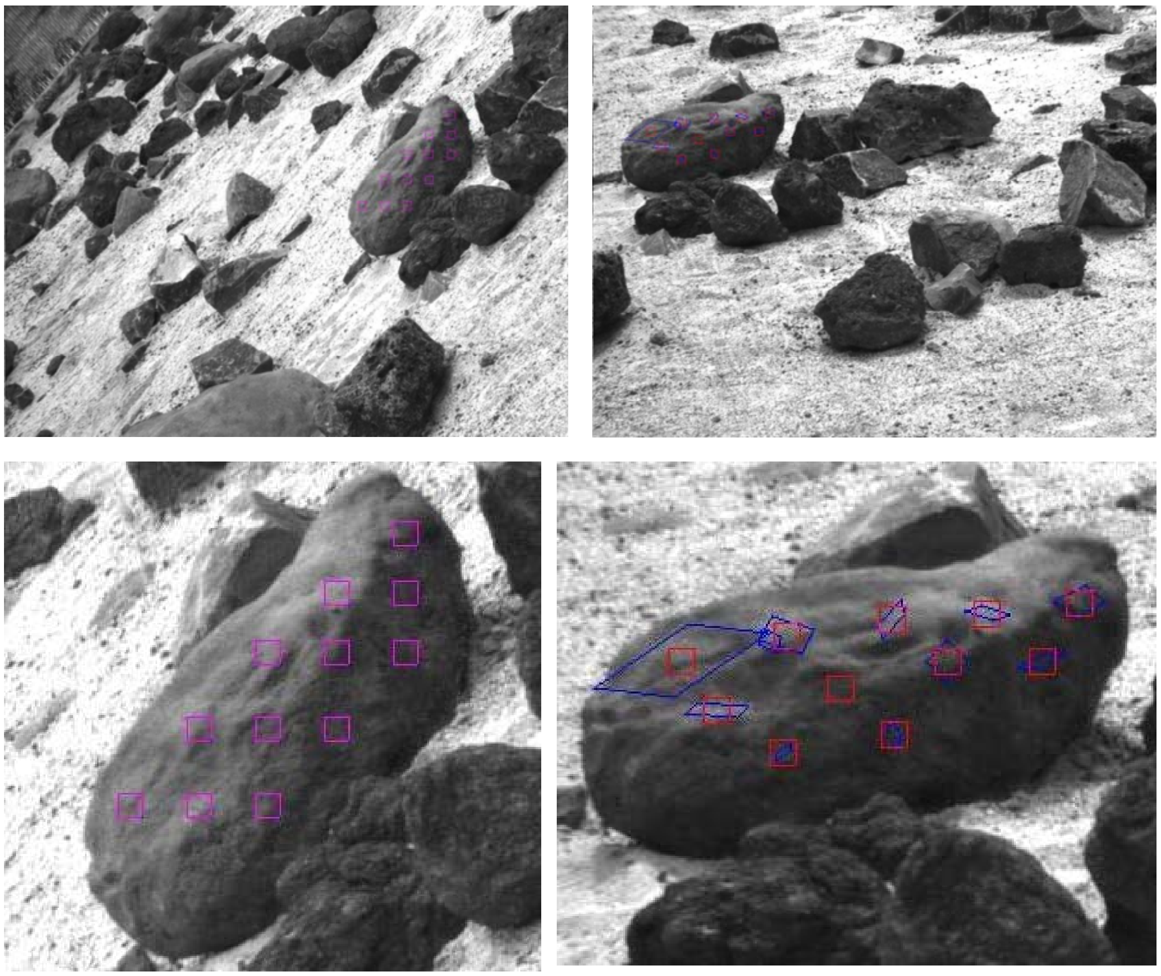


Figure 42. Beginning (top left) and end image after 40º roll (top right) using 15×15 window with 3 pyramid levels. Their close-up views are in bottom left and bottom right.

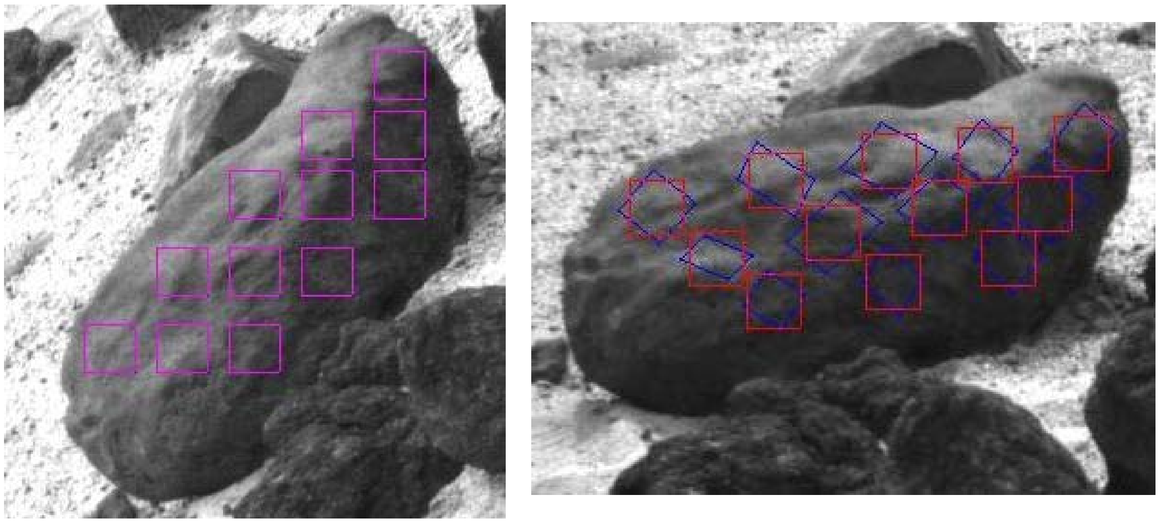


Figure 43. Beginning (top left) and end image after 40º roll (top right) using 15×15 window with 3 pyramid levels. Their close-up views are in bottom left and bottom right.

4.13 Yaw-Motion Tracking with Large-Image-Size Rocks

2-D tracking performance with yaw camera motion was similar to forward and roll camera motions. The 15×15 window with 4 pyramid levels is sometimes large enough to track targets selected from large-image-size rocks, while in other times larger image size windows were required. In Figure 44 for 8 mm lens, all 16 targets selected on large-image-size rocks, tracked well over the entire 45º yaw with 15×15 window with 4 pyramid levels. Examples of four enlarged target views at yaw 0º, 15º, 30º and 45º during tracking are shown in Figure 44.

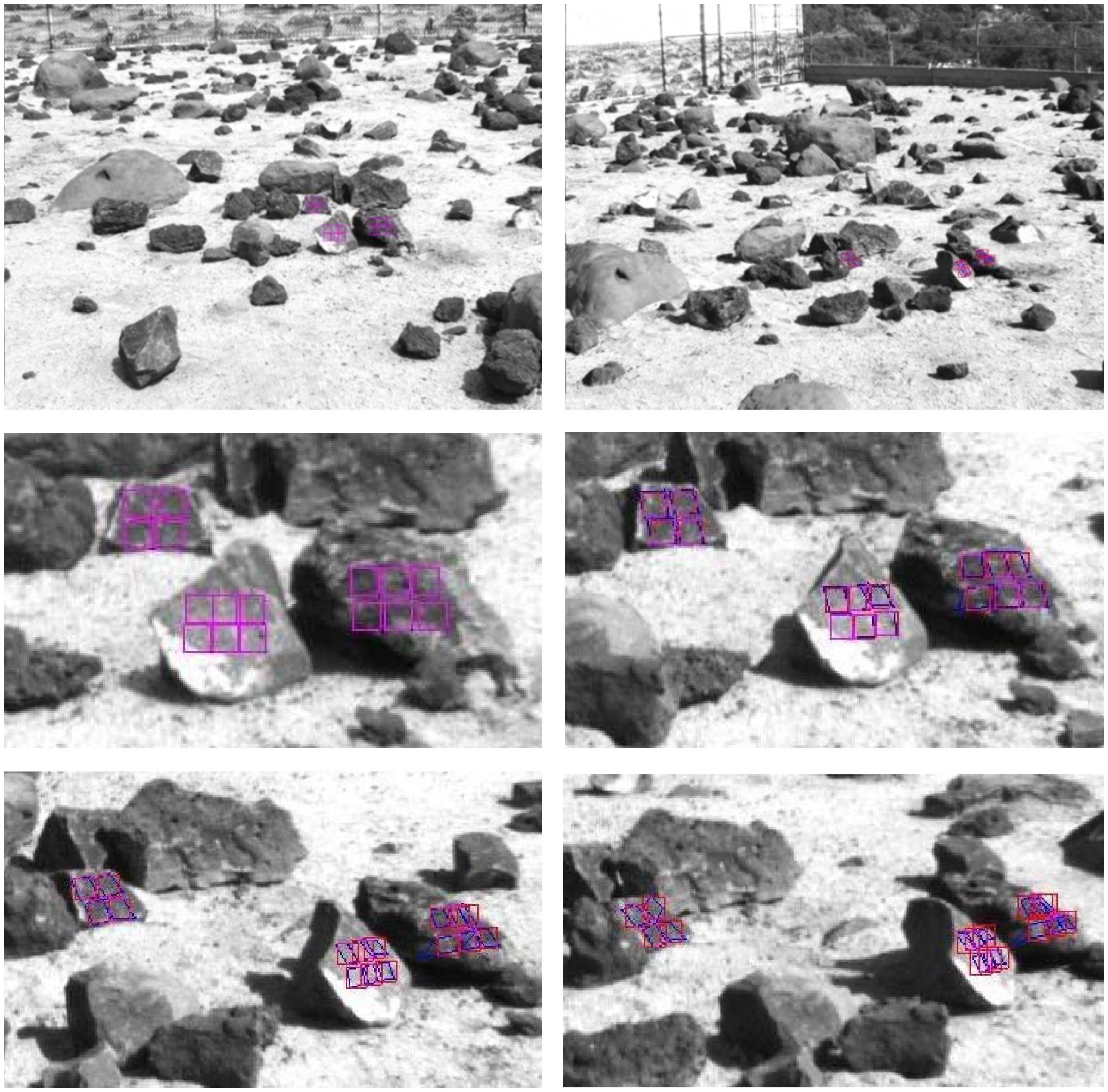


Figure 44. Beginning (top left), end (top right) images for 45º yaw camera motion with 8 mm lens. Enlarged close-up views show target tracking windows at yaw of 0º (middle left), 15º (middle right), 30º (bottom left) and 45º (bottom right).

In Figure 45 for 16 mm lens, 29×29 window with 3 pyramid levels did not provide good tracking performance. For yaw camera motions, images happened to have large image displacements, some of which were more than 30 pixels. To handle large image displacements, the number of pyramid levels was raised to 4. Tracking performance using 29×29 window with 4 pyramid levels (Figure 45) provided good tracking performance except two affine transform windows were a little off. We also ran 59×59 and 75×75 windows with 3 pyramid levels, both of which showed good tracking performance with reasonable affine transformed windows.

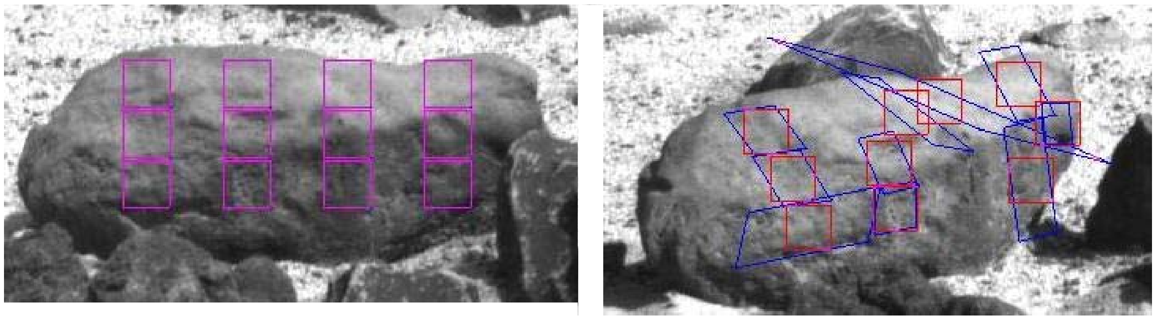


Figure 45. Begin/end images after 45º yaw motion using 29×29 window with 3 pyramid levels

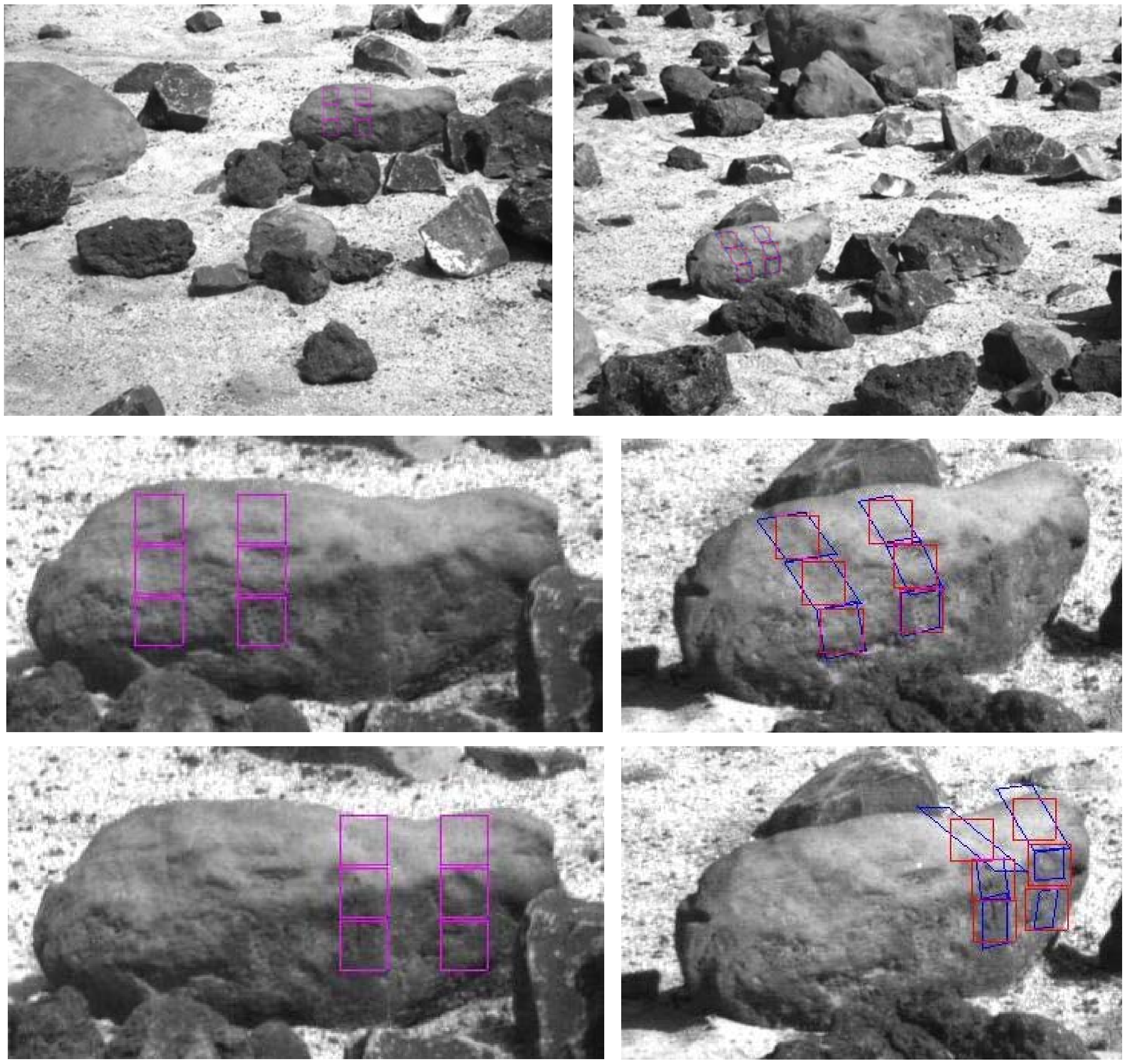


Figure 46. Begin/end images for 45º yaw motion using 29×29 window with 4 pyramid levels**:** full images (top) and enlarged views for six targets (middle) and for six others (bottom)

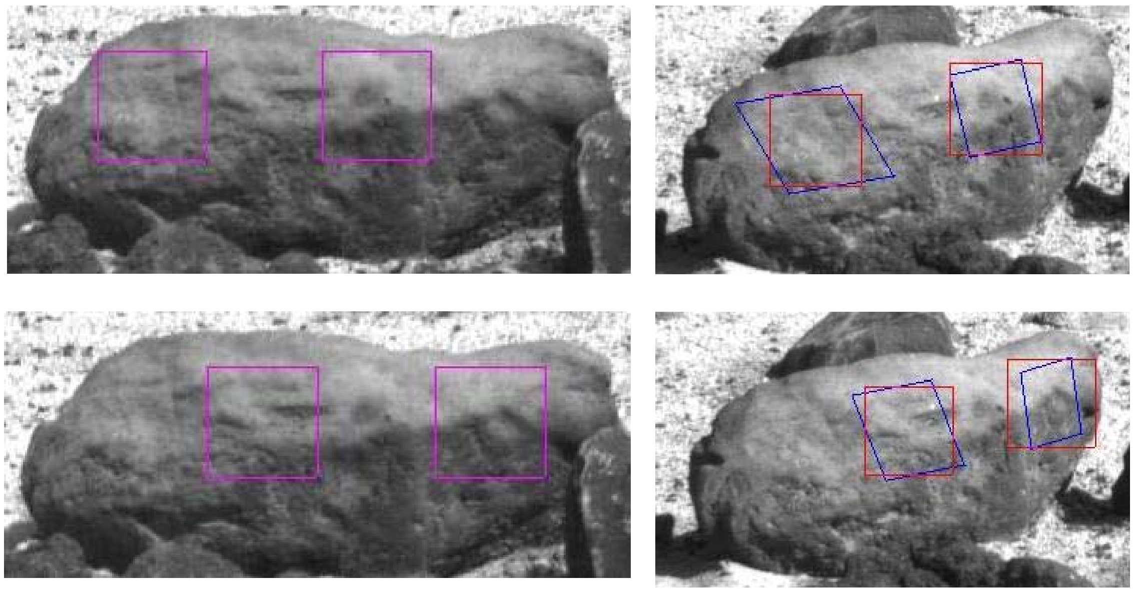


Figure 47. Begin/end images for 45º yaw motion using 75×75 window with 3 pyramid levels for two targets (top) and for two others (bottom)

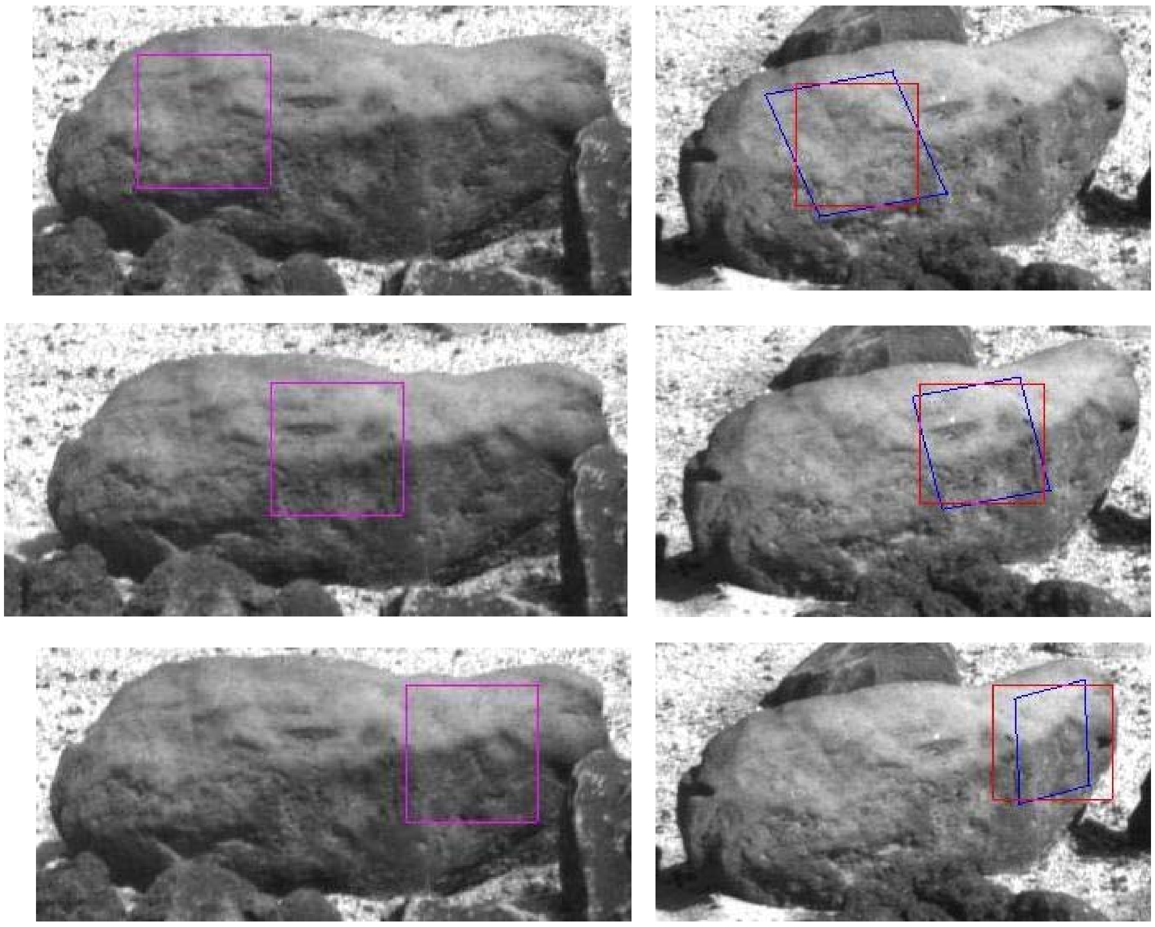


Figure 48. Begin/end images for 45º yaw motion using 75×75 window with 3 pyramid levels for three different targets (top, middle, and bottom)

2-D tracking performance for 16 mm lens was further tested with another large-image-size rock. For both 15×15 and 29×29 windows with 4 pyramid levels, only 6 targets out of 9 tracked in Figure 49. Top three targets were lost, and we noted that the background above the top occluding boundary of the rock changed from white to black during the course of tracking, because the very dark shadow of another rock shifted as the camera yaw changes.

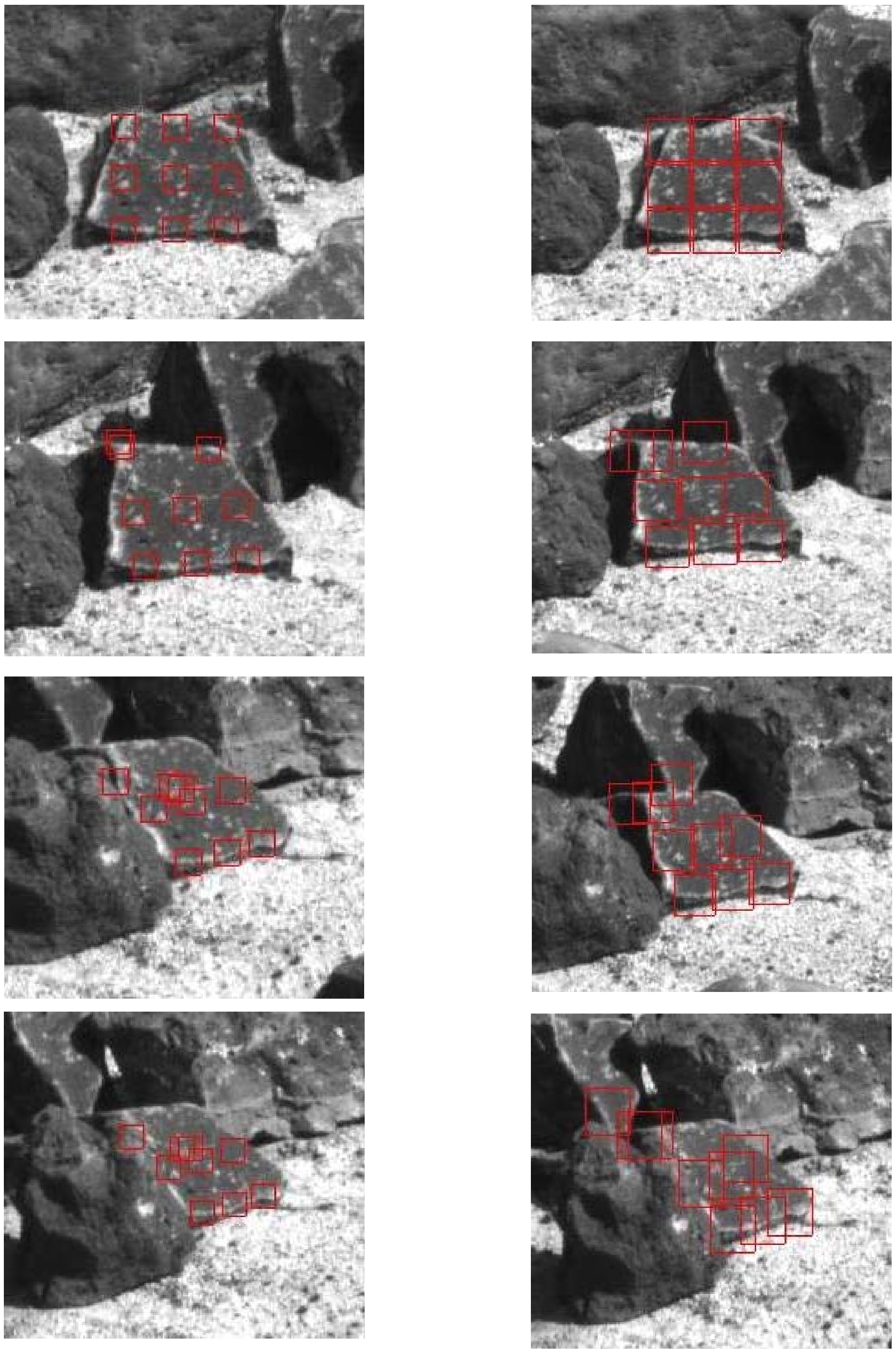


Figure 49. Enlarged close-up views showing target tracking windows at yaw of 0º (top row), 15º (second row), 30º (third row) and 45º (bottom row) for 45º yaw camera motion with 16 mm lens: 15×15 window (left column) and 29×29 window (right column) with 4 pyramid levels.

To avoid the dramatic background change, we selected the target so that it just fits the rock using a 75×75 window as shown in Figure 50. However, both 1 and 2 pyramid levels with 75×75 window showed poor tracking performance. Only when the number of pyramid levels increased to 3, it showed good tracking performance. It is not clear why 75×75 window with 2 pyramid levels did not track very well.

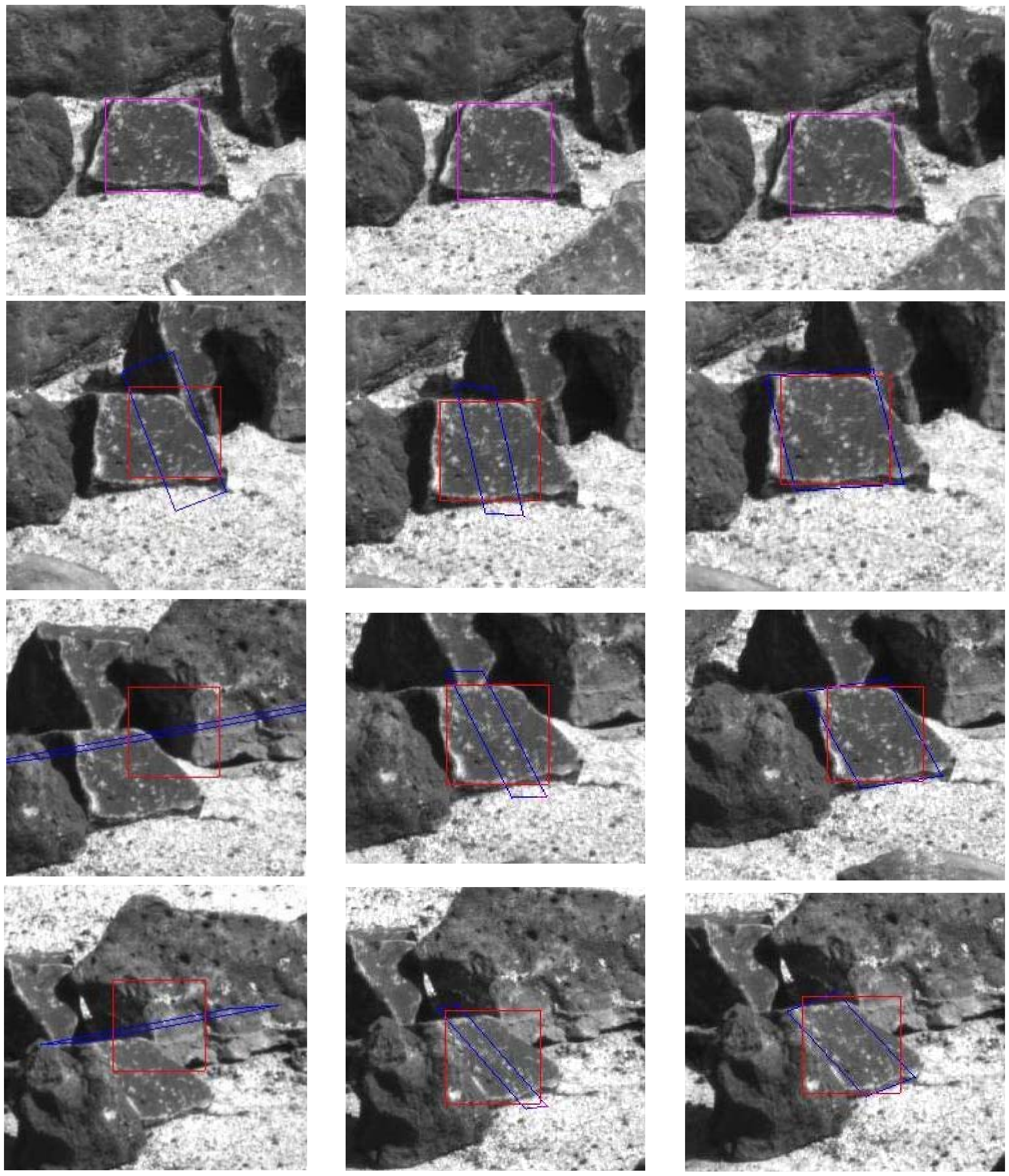


Figure 50. Enlarged close-up views showing target tracking windows at yaw of 0º (top row), 15º (second row), 30º (third row) and 45º (bottom row) for 45º yaw camera motion with 16 mm lens: 75×75 window with 1 (left column), 2 (middle column), and 3 pyramid levels (right column).

4.14 Alternate Template Update Method

In the combined translation and affine transform configuration that we used in this report, the affine template was updated every time the translation template was updated. An alternate update method was suggested, where only the translation template was updated while the affine template was never updated. The two template update methods were compared. For an example run with forward motion (Figure 51), tracking performance using the method updating translation template only was similar to that using the alternate method updating both translation and affine templates.

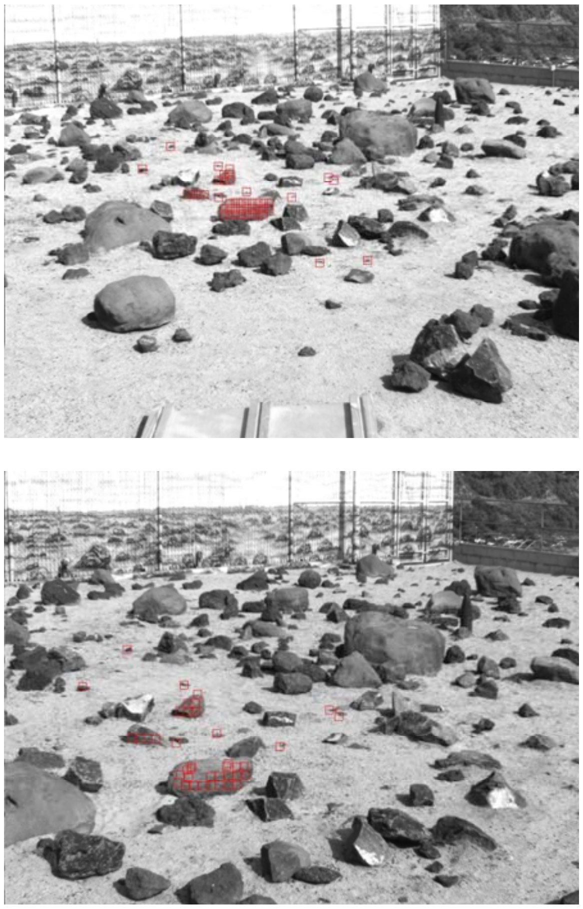
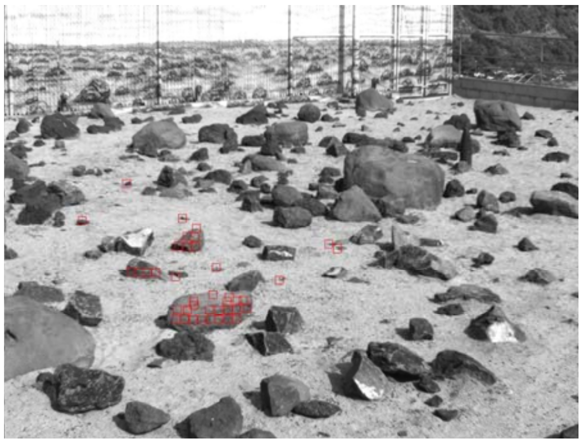



Figure 51. Affine template update vs. no affine template update: beginning image (top), end image resulted from the method updating both translation and affine templates (left), and end image resulted from the method updating translation template only (right).

For a different example run with yaw motion (Figure 52), however, the method updating both translation and affine templates showed more reliable tracking without losing the target. The alternate method updating translation template only lost the target near the end. However, it tended to yield more accurate affine transformed window (blue) when it tracked. In the alternate method, it turned out that the ssd (square-sum-of-differences) value for affine matching kept increasing because the affine template was never updated, while the ssd value for 2-D translation matching reduced to near zero whenever the template was updated. In the current version of the combined configuration, the affine matching correction occurs only when its ssd value is lower than the ssd value of the 2-D translation matching. Thus, in the alternative method updating the translation template only, affine matching is never in use except for the very beginning portion of tracking, indicating that this alternative is not a good option.

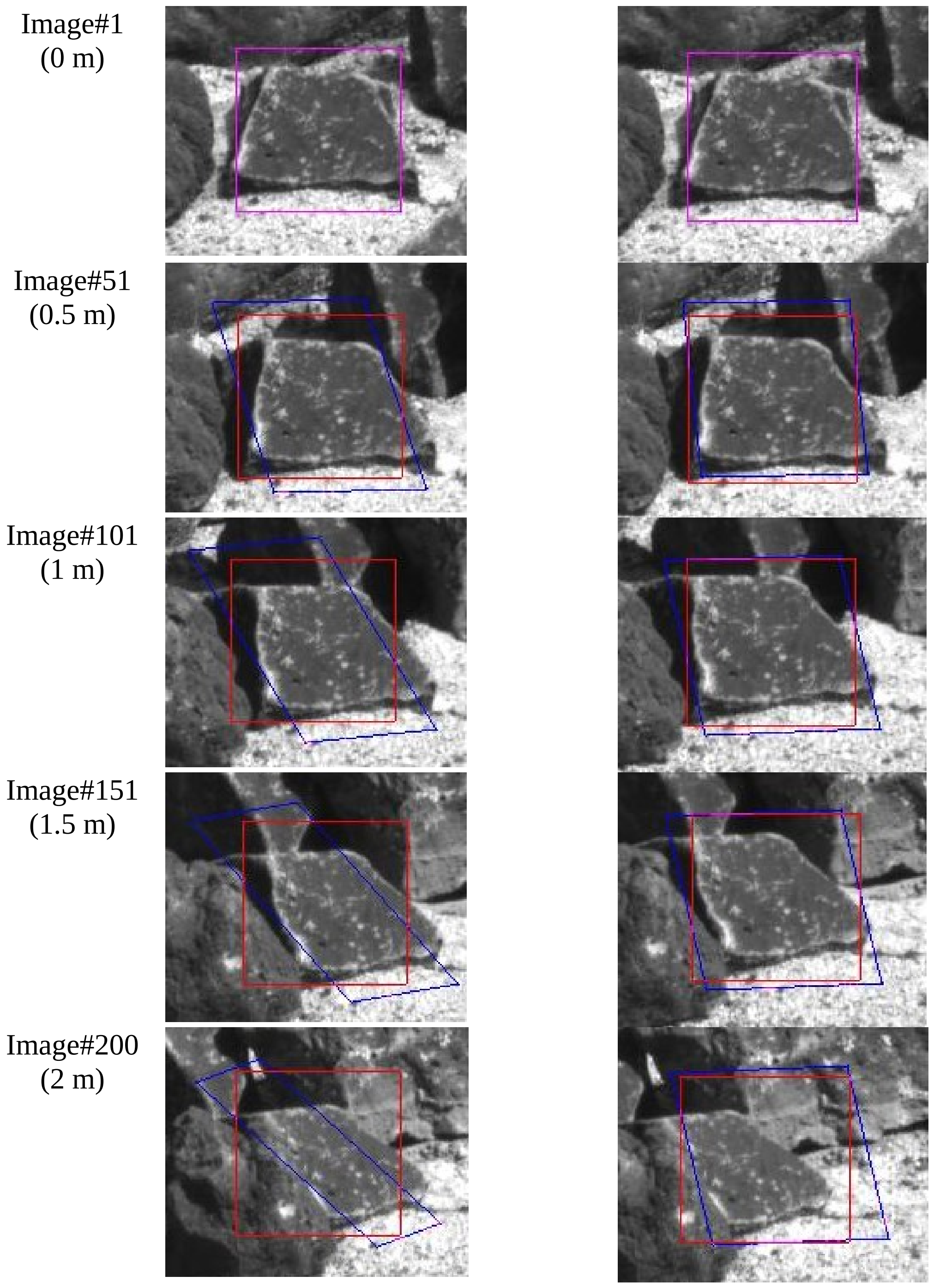


Figure 52. Comparing two update methods for tracking a target with yaw camera motion: the method of updating both translation and an affine templates (left column) and alternate method of updating translation template only (right column)

4.15 Improving tracking reliability

Here is a list of items to be considered to improve tracking reliability

1. Active camera control with 2-D/3-D tracking
2. Brute-force cross-correlation 2-D translation global matching in addition to the current iterative search
3. Instead of periodic template update by a fixed travel distance, use template update triggered by the thresholds of distance percent change, orientation change, and affine matching ssd.
4. Use an off-centered window, where target is not at the center of the window in case when a large window is needed, for example, to track a target without much texture on a surface of a large-image-size rock. It is best not to include the background beyond the occluding boundary of the rock. The off-center target position relative to the center of the tracking window must be computed.
5. Selective region matching. For example, 1) use a quadrilateral window of general shape rather than an upright square window, 2) use a combination of several small target windows, or 3) use a mask for planar surface of the rock if stereo range data available.
6. Automatic generation of the off-centered, selective-region target window when a target is selected. The target may be low-textured and near the occluding bouindary.
7. Cross-correlation matching or a Difference of Gaussian (DoG) filter to cope with ambient lighting changes
8. Advanced algorithm that detects target drift and loss reliably.

5. Software Bug Findings and Fixes

As shown in Figure 53, tracking performance with the initial delivery software was poor (left-side two plots). The final delivery software after first three bug fixes (right-side two plots) showed dramatic performance improvements.

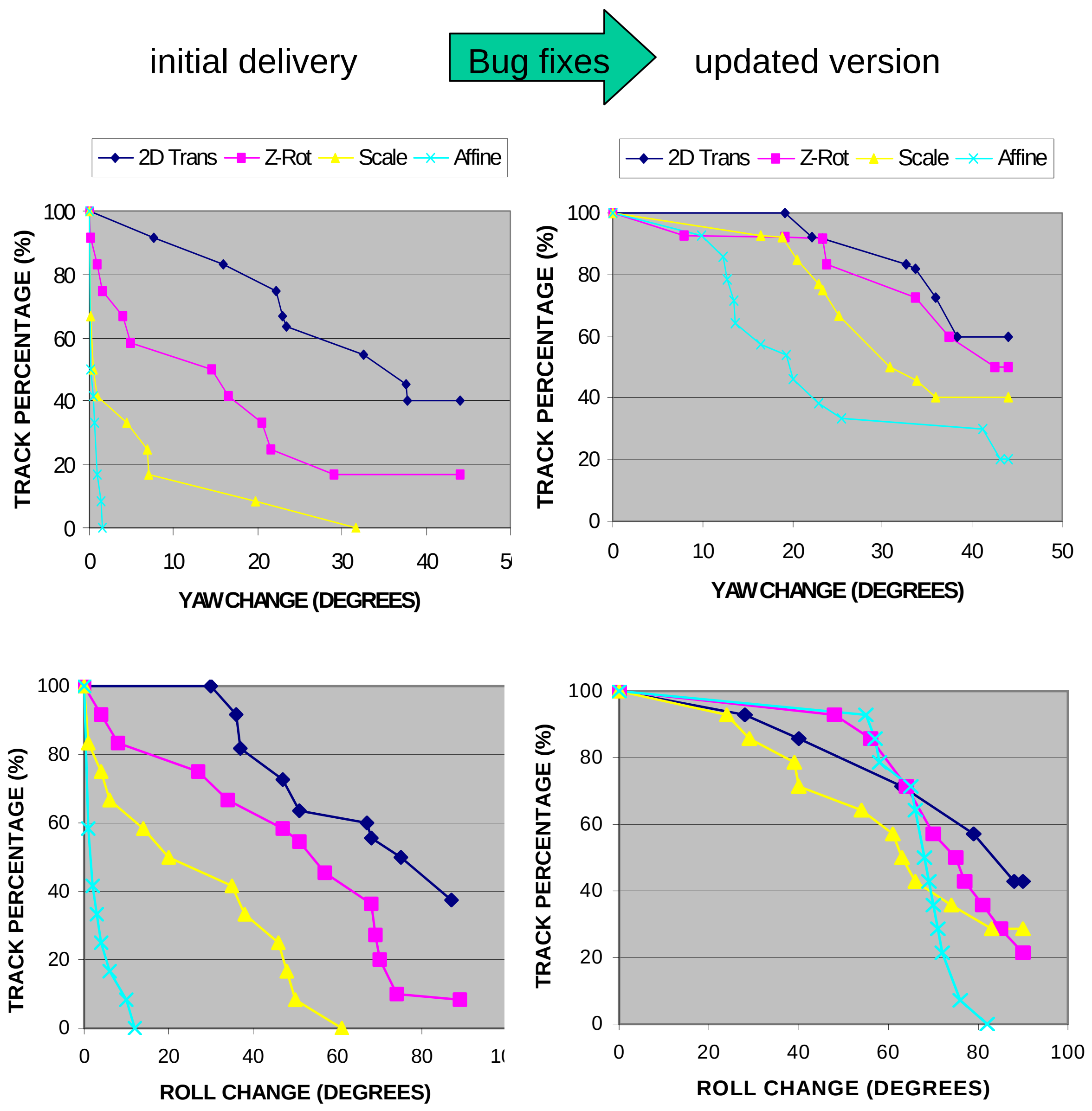


Figure 53. Tracking performance of different transforms for yaw (top row) and roll (bottom row) camera motions: before (left column) and after bug fixes (right column).

In collaboration with technology providers, we identified the following five software bugs.

1. **Partial derivatives inverse matrices at different pyramid levels.** Partial derivatives inverse matrices were not retrieved correctly at lower pyramid levels. This screwed up feature matching at lower pyramid levels.
2. **Delta translation updates between pyramid levels.** The delta translation updates were not halved when the pyramid level for matching went down, and not doubled when it went up. This also screwed up feature matching at lower pyramid levels.
3. **Interchange of width and height**. In two places, image width and height were swapped. This prevented tracking beyond the 768-pixel column of the 1024-column image.
4. **Crash after long runs with a large number of targets.** There are memory leaks in Feature_Window and Image_Pyramid classes. Objects created with new() during class constructions could not be deleted upon class destructions. This bug is not fixed yet.
5. **Crash for track windows at the image boundary**. The program crashes when target windows are at the image boundary. This bug is not fixed yet.

Among these five bugs identified, the first three were fixed in the latest delivery of the 2-D tracker software. The last two bugs do not directly affect the tracking performance. However, sometimes we could not track several targets in a single run when the number of targets, window size, and/or the number of pyramid levels is large. In this case, we had to divide the targets and execute 2-D tracker in several runs as shown, for example, in Figures 47 and 48.

6. Conclusion

We evaluated the target tracking performance using the JPL 2-D target tracking software. After critical bug fixes, the latest version of the software demonstrated relatively good tracking performance of average 80% to 100% success rate when 15×15 window with 3 or 4 pyramid levels are used for 4-m straightforward, 90º roll, and 45º yaw camera motions. Further detailed examination revealed three main causes of tracking failures: 1) large image displacements, 2) low texture, and 3) occlusion with foreground/background change. Regarding these issues, our experiments found the following. A larger window size with larger pyramid levels has a larger effective window size and thus helps cope with larger image displacements. Targets selected on large-image-size rocks tend to have low texture, and a larger window size greatly improves the tracking performance to near 100% tracking as long as the objects within the target window are approximately planar. A larger window is a great solution to produce more reliable tracking for low-texture targets, but introduces three additional issues: 1) larger drift error, 2) off-centered target position relative to the target window, and 3) inclusion of non-planar objects including the background beyond the occluding boundary. More conclusive experiments including full 10-m forward motion tests will be performed when the 2-D/3-D tracking software with active camera control is available.

References

M. Bajracharya, I. Nesnas, CLARAty Delivery for 2-D Visual Tracking, an html document, April 2003.

T. Estlin, R. Volpe, I. Nesnas, D. Mutz, F. Fisher, B. Engelhardt, S. Chien, "Decision-Making in a Robotic Architecture for Autonomy." Proceedings of 6th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS), Montreal Canada, June 18-21 2001.

W. Kim, R. Steinke, R. Steele, and A. Ansar, Camera Calibration and Stereo Vision Technology Validation Report, JPL D-27015, June 2003.

I. Nesnas, R. Volpe, T. Estlin, H. Das, R. Petras D. Mutz, "Toward Developing Reusable Software Components for Robotic Applications" Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Maui Hawaii, Oct. 29 - Nov. 3 2001.

A. Steltzner, “MSL Baseline Description Document,” 2003.

L. Tamppari, email communications, August 2003.

R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, H. Das, "The CLARAty Architecture for Robotic Autonomy." Proceedings of the 2001 IEEE Aerospace Conference, Big Sky Montana, March 10-17 2001.

R. Volpe, I.A.D. Nesnas, T. Estlin, D. Mutz, R. Petras, H. Das, "CLARAty: Coupled Layer Architecture for Robotic Autonomy." JPL Technical Report D-19975, Dec 2000.